UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Sammy Johnatan Carbajal Ipenza

Efficient Pulse-Density Modulated Microphone Array Processing

Processamento Eficiente de Arranjos de Microfones Modulados em Densidade de Pulso

Campinas
2020

Sammy Johnatan Carbajal Ipenza

Efficient Pulse-Density Modulated Microphone Array Processing

Processamento Eficiente de Arranjos de Microfones Modulados em Densidade de Pulso

Dissertation presented to the School of Electrical and Computer Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Electrical Engineering, in the area of Telecommunications and Telematics.

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Concentração de Telecomunicações e Telemática.

Orientador: Bruno Sanches Masiero

Este exemplar corresponde à versão final da dissertação defendida pelo aluno Sammy Johnatan Carbajal Ipenza, e orientada pelo Prof. Dr. Bruno Sanches Masiero.

Campinas
2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

C177e

Carbajal Ipenza, Sammy Johnatan, 1989-
Efficient pulse-density modulated microphone array processing / Sammy Johnatan Carbajal Ipenza. – Campinas, SP : [s.n.], 2020.

Orientador: Bruno Sanches Masiero.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Microfones. 2. Modulação de pulso (Eletrônica). 3. Impedância acústica. 4. Modulçao de duração de pulso. I. Masiero, Bruno Sanches, 1981-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Processamento eficiente de arranjos de microfones modulados em densidade de pulso
Palavras-chave em inglês:
Microphone
Pulse modulation (Electronics)
Acoustic impedance
Pulse-width modulation
Área de concentração: Telecomunicações e Telemática
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Bruno Sanches Masiero [Orientador]
Vitor Heloiz Nascimento
Renato da Rocha Lopez
Data de defesa: 30-11-2020
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-4150-9044
- Currículo Lattes do autor: http://lattes.cnpq.br/9757083414604365

Comissão Julgadora – Dissertação de Mestrado

Candidato: Sammy Johnatan Carbajal Ipenza **RA:** 159575

Data da defesa: 30 de Novembro de 2020

Título da Tese: "Efficient Pulse-Density Modulated Microphone Array Processing"

Prof. Dr. Bruno Sanches Masiero (Presidente, FEEC/UNICAMP)
Prof. Dr. Vitor Heloiz Nascimento (USP)
Prof. Dr. Renato da Rocha Lopes (FEEC/UNICAMP)

A ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de Pós-Graduação da Faculdade de Engenharia Elétrica e de Computação.

A mis padres, Juan y Celia; cuyos ejemplos
de amor, dedicación y perseverancia,
como faroles, siempre guían mi camino.

Acknowledgments

First, I would like to thank to the Campinas State University and specially to my supervisor Bruno Masiero, for believing in me during these two years, for his guidance, support and, foremost, his patience during the development of this work. I am also grateful to NXP Semiconductors, specially to my manager Mauricio Brochi, for giving me the flexibility to setup my schedule according to my academics activities; and to Felipe Ponce for his assistance in some laboratory testings.

Also, I want to thank foremost to my parents, Juan Carbajal and Celia Ipenza, and to my brothers, Juan Carlos and Jorge, for the constant support, encouragement and motivation along the different personal and professional projects of my life. Also I want to thank to my uncle Wilder, who in my childhood help me to overcome my fear to maths and settled the roots to me pursue this engineering path.

Finally, but not less important, I want to thank to my girlfriend Jaqueline Torres for her love, support and patience; and to my little pet Ivar whom, unaware of it, was the best companion during all my nights without sleep writing this thesis.

Abstract

Nowadays, pulse-density modulated (PDM) digital microphones are widely used on commercial applications as they have become a popular way to deliver audio to digital processors on mobile applications. However, as state-of-the-art array processing algorithms assume that all microphone signals are available in pulse-code modulated (PCM) representation, these digital microphones require costly high-order decimation filters to translate PDM bitstreams to baseband multi-bit PCM signals. In that manner, the implementation of microphone array algorithms in embedded systems, where processing resources are critical, or in very large-scale integration (VLSI) circuits, where power and area are critical, may become very expensive because of the use of the tens of decimation filters required to convert PDM bitstreams into PCM signals. To overcome these limitations, this dissertation explores and proposes resource-efficient methods to implement microphone array beamformers using PDM microphones. In this sense, after to thorough review of the state-of-the-art decimation filter design methods and the state-of-the-art beamforming implementation methods, it proposes: an algorithm to design decimation filters with minimum number of additions per second; efficient beamforming implementations that work on the PDM domain; and a novel beamforming architecture that fuses both delay and decimation operations based on maximally flat (MAXFLAT) filters. It shows that the beamformers implemented on the PDM domain have a good resource-efficiency as requires less additions per second and memory elements than other conventional methods. Finally, it concludes that the proposed MAXFLAT-based approach has the best trade-off between storage and computational efficiency in comparison to state-of-the-art and other proposed PDM domain implementations.

Resumo

Atualmente, os microfones digitais modulados por densidade de pulso (PDM) são amplamente utilizados em aplicações comerciais, já que esta é uma maneira eficiente de transmitir informação de áudio para processadores digitais em dispositivos móveis. No entanto, como o estado-da-arte em algoritmos de processamento digital de arranjos assume que todos os sinais recebidos dos microfones estão em uma representação em banda-base, estes microfones digitais requerem custosos filtros de decimação de alta ordem para converter o fluxo PDM para a modulação por código de pulso (PCM) em banda-base. Assim, a implementação destes algoritmos em sistemas embarcados, onde os recursos de processamento são críticos, ou em circuitos integrados (VLSI), onde a energia consumida e área também são críticas, pode se tornar muito dispendiosa devido ao uso de dezenas de filtros de decimação para converter os sinais de PDM para PCM. Para superar essas limitações, essa dissertação explora e propõe métodos eficientes em recursos para a implementação de arranjo de microfones usando microfones digitais PDM. Com esse intuito, apos rever os atuais métodos de design de filtros de decimação e os atuais métodos de implementação de arranjos de microfones, propõe-se: um algoritmo para fazer o design de filtros de decimação com o mínimo número de adições por segundo, implementações eficientes de arranjos de microfones que trabalham no domínio do PDM, e um método eficiente para implementação de arranjos de microfones baseado em filtros maximamente planos (MAXFLAT). Demonstra-se que o filtro espacial implementado no domínio do PDM é mais eficiente em recursos porque precisa de menos adições por segundo e elementos de memoria que as implementações convencionais. Finalmente, conclui-se que a implementação baseada em filtros MAXFLAT tem um melhor compromisso entre requisitos de armazenamento e poder de computação que o estado-da-arte e os métodos no domínio do PDM propostos.

List of Figures

List of Tables

Symbols

$B_{N,K,d}(z)$ Samadi filter binomial component.

B_w bandwidth.

c sound speed.

c normalized passband frequency in a CIC compensator.

$N^{+}_{G_c}$ number of adders for CIC compensator.

Δf transition bandwidth.

Δf_j jth-stage transition bandwidth.

$\Delta \bar{f}$ normalized transition bandwidth.

$\Delta \bar{f}_j$ jth-stage normalized transition bandwidth.

$D_{\min}$ minimum distance between microphones.

δ_p^j jth-stage passband ripple.

$d_{\mathbf{lim}}$ physical limit of the d parameter.

Δ_m delay from the array center to the mth microphone.

$d_{\max}$ maximum allowed delay parameter.

$\Delta_{\max}$ maximum required delay.

Δ filter delay.

δ_p passband ripple.

δ_s stopband ripple.

d Samadi filter delay parameter.

δ_s^j jth-stage stopband ripple.

$N_j^{\mathbf{eqr}}$ optimal jth-stage minimum required filter length.

ρ_j jth-stage significance coefficient rate.

$f_{\mathbf{cpu}}$ estimated minimum frequency in a processor.

f_i input sampling rate.

f_j jth-stage output sampling rate.

f_{j-1} $j-1$th-stage output sampling rate.

$\frac{f}{f_{j-1}}$ jth-stage normalized frequency.

f_o output sampling rate.

F_p passband frequency.

F_p^j jth-stage passband frequency.

F_s stopband frequency.

F_s^j jth-stage stopband frequency.

α group delay.

$H_j(e^{2\pi i \frac{f}{f_{j-1}}})$ jth-stage low-pass filter impulse response.

$H_M(z)$ sigma-delta modulator feedback impulse response.

$H(e^{2\pi i \frac{f}{f_i}})$ overall low-pass filter impulse response.

$H_{N,K,d}(z)$ Samadi filter impulse response.

$H(e^{j\omega})$ low-pass filter impulse response.

$H(z)$ low-pass filter impulse response.

$S^z_{\mathbf{dec}}$ decimation filter's storage requirement.

K CIC filter order.

$S^z_{\mathbf{FFT}}$ FFT's storage requirement.

S_z storage requirements.

S^z_j jth-stage storage requirement.

S^z_J Jth-stage storage requirement.

K_j jth-stage CIC filter order.

$S^z_{\mathbf{bf}}$ beamformer's storage requirement.

K number of zeros at $z=-1$ in a Samadi filter.

$L_{\mathbf{acc}}$ filter accumulator length.

$L_{\mathbf{frame}}$ frame length (for frequency domain implementations).

$L_{\mathbf{in}}$ filter input length.

L_j jth-stage required filter output length.

L_{j-1} $j-1$th-stage required filter output.

$L_{\mathbf{out}}$ filter output length.

$\delta_j^{\mathbf{band}}$ jth-stage design stopband and passband ripples.

N_j optimal jth-stage minimum required filter length.

ρ_j jth-stage significance coefficient rate.

M number of microphones.

N minimum required FIR filter length.

N_I interpolation filter length.

N_j jth-stage minimum required FIR filter length.

$N_{\mathbf{nz}}$ number of nonzero coefficients in the filter impulse response.

N Samadi filter order.

ν roll-off factor.

R decimation factor.

$S^+_{\mathbf{dec}}$ decimation filter's number of additions per second.

$S^+_{\mathbf{FFT}}$ FFT's number of additions per second.

$T^+_{\mathbf{FPGA}}$ estimated number of adders in an FPGA running at 64 MHz.

S_+ number of additions per second.

S^+_j jth-stage number of additions per second.

S^+_J Jth-stage number of additions per second.

$T^+_{\mathbf{lp}}$ estimated number of adders in a VLSI circuit running at 10 MHz.

$S^+_{\mathbf{bf}}$ beamformer's number of additions per second.

ρ significance coefficient rate.

R_j jth-stage decimation factor.

$S^*_{\mathbf{dec}}$ decimation filter's number of multiplications per second.

$S^*_{\mathbf{FFT}}$ FFT's number of multiplications per second.

S_* number of multiplications per second.

S^*_j jth-stage number of multiplications per second.

S_J^* Jth-stage number of multiplications per second.

$S_{\mathbf{bf}}^*$ beamformer's number of multiplications per second.

$S_{\mathbf{dec}}^o$ decimation filter's total number of additions per second.

S_o total number of additions per second.

S_j^o jth-stage total number of additions per second.

S_J^o Jth-stage total number of additions per second.

$S_{\mathbf{bf}}^o$ beamformer's total number of additions per second.

single_direct single-stage direct form implementation.

single_eff single-stage efficient direct form implementation.

single_memsav single-stage memory-saving polyphase implementation.

U_p passband frequency range.

U_p^j jth-stage passband frequency range.

U_s stopband frequency range.

U_s^j jth-stage stopband frequency range.

V_p passband angular frequency range.

V_p^j jth-stage passband angular frequency range.

V_s stopband angular frequency range.

V_s^j jth-stage stopband angular frequency range.

ω_c cutoff frequency.

w_m mth-filter channel gain.

ω_p angular passband frequency.

ω_p^j jth-stage angular passband frequency.

ω_s angular stopband frequency.

ω_s^j jth-stage angular stop frequency.

$y_j[n]$ jth-stage filter output.

Acronyms

ΣΔ sigma-delta.

ΣΔM ΣΔ modulator.

ADC analog-to-digital converter.

APS additions per second.

CIC Cascade Integrator-Comb.

CSD canonical signed digit.

DAS delay-and-sum.

DFT Discrete Fourier transform.

DoA direction of arrival.

DR dynamic range.

FFT Fast Fourier transform.

FIR Finite Impulse Response.

FPGA Field Programmable Gate Array.

IFFT Inverse Fast Fourier transform.

IIR Infinite Impulse Response.

IoT internet of things.

IVA intelligent virtual assistants.

LPF low-pass filter.

MAXFLAT maximally flat.

MEMS micro-electro-mechanical system.

MPS multiplications per second.

OSR oversampling rate.

PAPS PDM-mic array processing system.

PCM pulse-code modulated.

PDM pulse-density modulated.

PDM-mic PDM microphone.

PSO particle swarm optimization.

SNR signal-to-noise ratio.

THD total harmonic distortion.

THD+N total harmonic distortion plus noise.

VLSI very large-scale integration.

Contents

Chapter 1

Introduction

1.1 Problem Statement

In the last decades, sensor array processing emerged as an active area of research in the estimation of space-time parameters. Array processing applications are applied to resolve many real-world problems. In telecommunications, antenna arrays are steered to one user direction to reduce interference between users. Radar and sonar use antennas and hydrophones arrays respectively to calculate parameters like direction of arrival (DoA), velocity and range. In medicine, sensor arrays are used for medical imaging, and planar biomagnetic sensor arrays are used in electrocardiograms to localize brain activity. In industry, sensor arrays are used in automatic monitoring and fault detection [4].

Recently, microphone array processing has emerged to resolve problems concerning internet of things (IoT) applications and hands-free communications, as this kind of communication becomes a standard option in many consumer devices like mobile phones, speakerphones and smart speakers, which are broadly used in conference rooms, desktop devices, and intelligent virtual assistants (IVA) in both consumer and industrial devices.

However, due to the complex characteristics of speech signals (non-static source, intermittent and broadband) and the usual environmental conditions (reverberation, and non-stationary additive noise); microphone array implementation is still costly and requires many microphones. For instance, consider a microphone that has a given signal-to-noise ratio (SNR) value at 2 cm from the talker; to attain the same SNR at 10 cm from the talker will require an array of 5 microphones. In the same way, if the required distance increases to 1 m or 2 m, the array will require 50 or 100 microphones respectively to attain the same SNR [5]. This large number of microphones increases the aperture size that is already constrained to: 1 cm diameter for hearing aids, 5 cm for automotive and 10 cm for consumer desktop devices [6]. In addition, each extra microphone in the design would require new routing, new placement conditions and more processing resources, which will

increase the system cost and power consumption, a critical factor for IoT and mobile applications.

Digital micro-electro-mechanical system (MEMS) microphones introduced in 2006 [7] have emerged as an alternative to overcome the array aperture size and cost limitations. As these microphones have an analog-to-digital converter (ADC) incorporated as pre-amplifier, they have a single line pulse-density modulated (PDM) output; because of that they are also known as PDM microphones (PDM-mics). A decimation filter (also known as PDM-to-PCM converter) demodulates this PDM bitstream output to a pulse-code modulated (PCM) signal. Unfortunately, the implementation of this decimation filter is still not cheap, as its cost (measured in die area and power) increases with the quality of the desired audio signal. Take for example the case of a microphone array using these PDM-mics like the delay-and-sum (DAS) beamformer in Figure 1.1. This architecture requires a decimation filter for each microphone input, so that the implementation cost and power consumption will increase proportionally with the number of microphones, being even more expensive for practical applications.

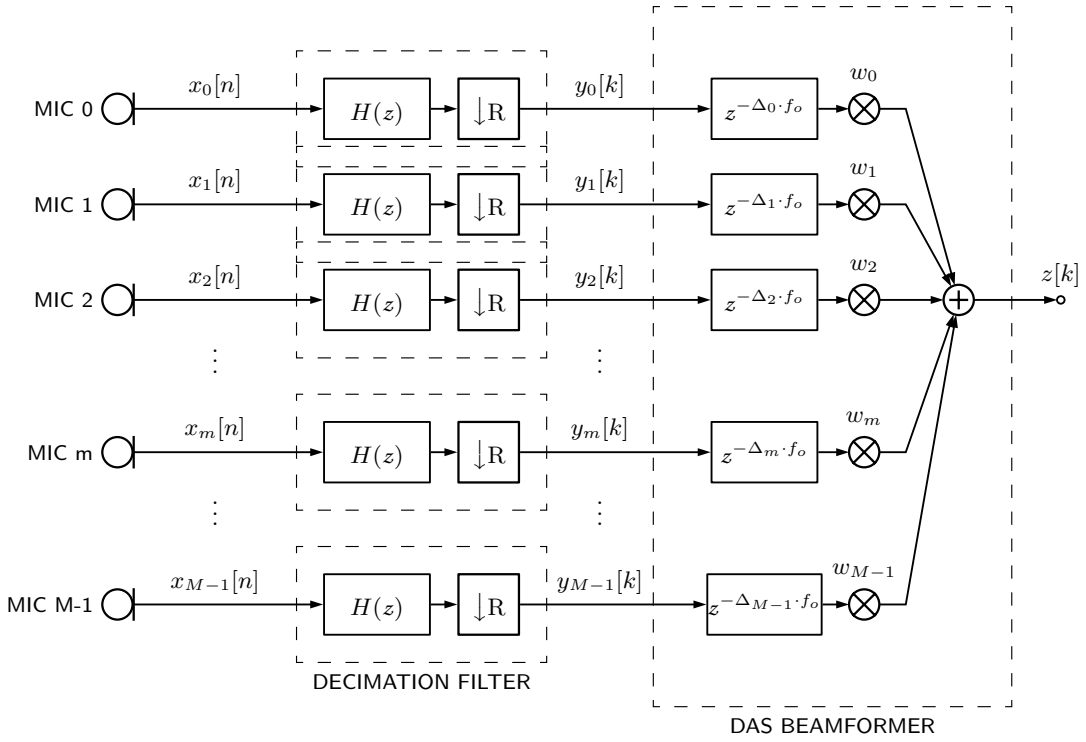


Figure 1.1: PDM-mic array DAS beamformer

1.2 Objectives

As one of the main limitations of the use of PDM-mic arrays and beamformers with many microphones is still the implementation costs, the main objective of this thesis is:

"Given a decimation filter specification, find resource-efficient beamformer implementation architectures using PDM-mics."

To achieve its main objective, this thesis also formulates three secondary objectives. These objectives are different approaches that try to attain the main objective, each with its advantages and disadvantages to be used in different designs and implementation constraints, all based on previous experiences and related works in filter processing.

1.2.1 Objective 1

"Given a decimation filter specification, find an algorithm to determine the most efficient decimation filter architecture."

The decimation filter or PCM-to-PDM converter is the basis of any PDM-mic interface. Many architectures are used in state-of-the-art implementations, most of them based on Cascade Integrator-Comb (CIC) filters, as they do not require multipliers on their implementation. However, to the best of the author's knowledge, there is not any study or algorithm that allows one to determine the most efficient decimation filter architecture given a desired filter specification.

In this way, this work proposes an algorithm to design an efficient decimation filter with minimal resources. The results of this algorithm are meant to be used in any application concerning PDM, not only with PDM-mics. Nevertheless, as the focus of this thesis is to apply the algorithm results in beamforming implementations, the algorithm is analyzed by comparing the implementation costs of PCM domain (Figure 1.1) and PDM domain beamformers (Figure 1.2).

1.2.2 Objective 2

"Given a decimation filter and a beamformer specification, find an efficient beamforming implementation that works on the PDM domain."

A new beamformer architecture, shown in Figure 1.2, is proposed and we show that it is possible to perform beamforming on the PDM domain i.e., before the PDM-to-PCM converter. We also show that a frequency domain implementation does not provide any gain in resource usage because of the large memory and logic requirements to implement the Fast Fourier transform (FFT) blocks.

While the same referred work [8] focuses its analysis in frequency domain implementations only, the present work analyzes also time domain implementations, and it shows that time domain implementations are resource-efficient.

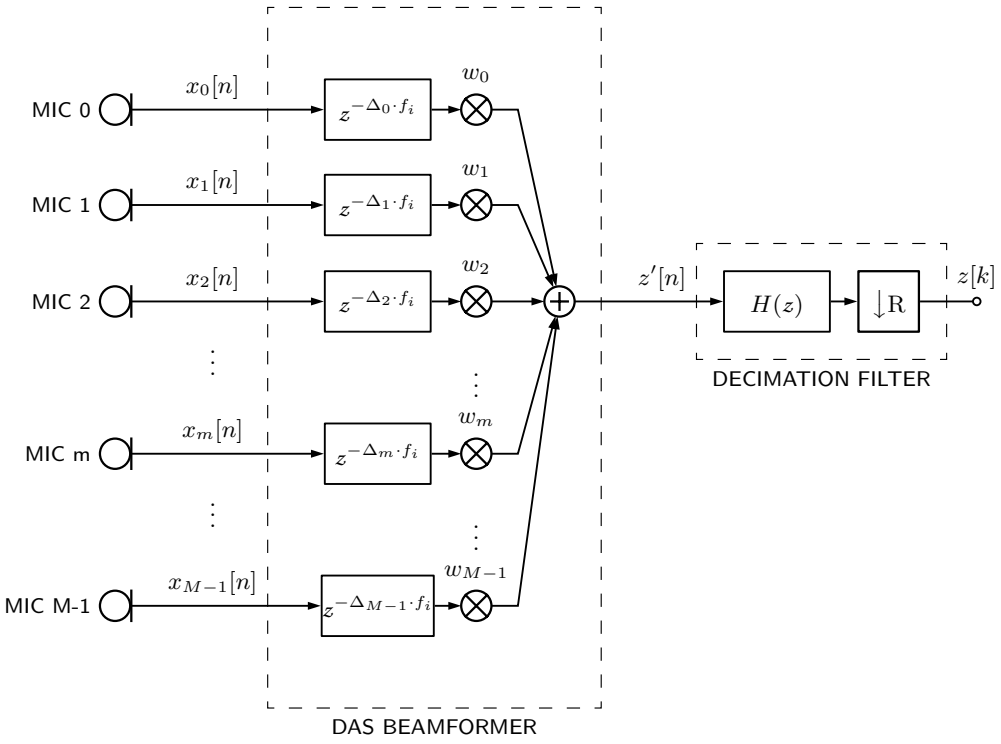


Figure 1.2: PDM-mic array DAS beamformer at PDM domain

1.2.3 Objective 3

"Given a decimation filter and a beamformer specification, find an architecture that fuses both delay and decimation operations."

As shown in Figure 1.1, a beamformer with PDM-mics is the conjunction of two fundamental operations: delay and decimation. Both operations are frequently performed separately in state-of-the-art methods, i.e., without any resource sharing between them, which results in suboptimal architectures in most of the cases. One architecture that fuses delay and filtering operations using a maximally flat (MAXFLAT) filter architecture was proposed by [9], and will be referred here as *Samadi filter*.

This thesis proposes: (1) a new decimation filter with embedded delay, dubbed *delayed decimation filter*, which is based on Samadi's filter; and (2) a beamformer architecture based on the delayed decimation filter as shown in Figure 1.3. Finally, we show that the proposed architecture is resource-efficient.

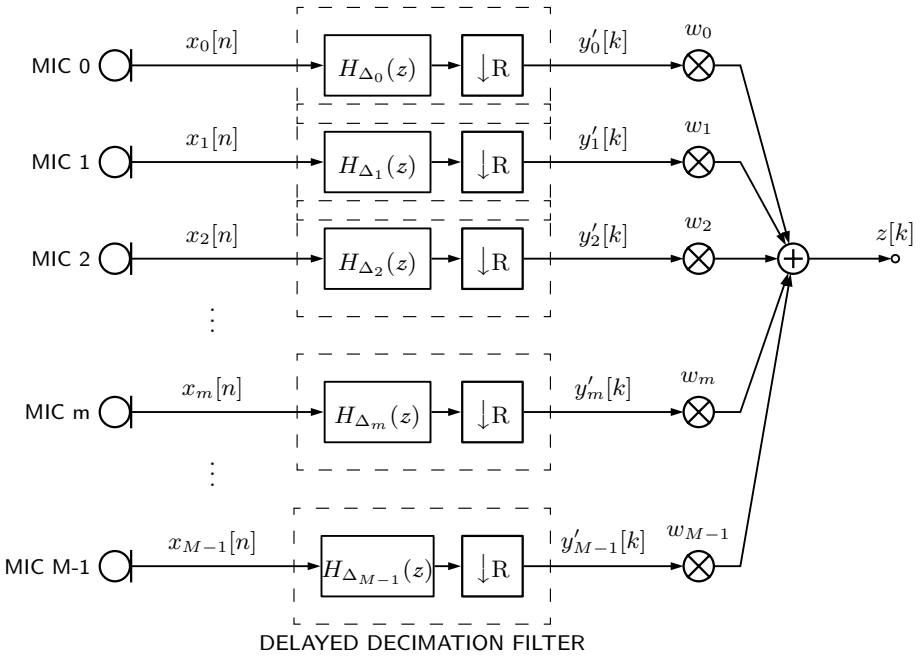


Figure 1.3: PDM-mic array DAS beamformer using delayed decimation filters

1.3 Contributions

The main contributions of this dissertation are the following:

1. It proposes a universal multi-stage filter design method optimized for the decimation filter's total number of additions per second (S^o_{dec}). This method also includes the following specific contributions:

   (a) A new method to compensate CIC filters for angular passband frequency $\omega_p \leq \frac{\pi}{4R}$.

   (b) A generalized method to design CIC filters.

2. It analyzes the resources required to implement DAS beamformers at PCM and PDM domains.

3. It proposes an efficient DAS beamformer architecture based on *delayed decimation filters*. This proposal also includes the following contribution:

   (a) Review of Samadi's filters and proposal of a method to design decimation filters based on it.

1.4 Metrics

In order to measure the efficiency of the proposed algorithms, methods and structures of this work, we need to establish some adequate metrics to compare them

quantitatively to the state of the art methods. As this work is focused in reducing the implementation costs, or in other words, in reducing the resource requirements, the usage of these requirements can be quantified by the number of multiplications per second (S_*), the number of additions per second (S_+) and the storage requirements (S_z); measured in multiplications per second (MPS), additions per second (APS) and number of bits respectively.

Also, assume that a multiplier has two operands $op1$ and $op2$ of L-bits width, such that $op2 = \overline{b_{L-1}b_{L-2}\dots b_1b_0}$, where b_i is the ith bit in a twos-complement representation. A multiplication operation can be expanded in a series of additions and binary shifts as follows

$$\begin{aligned} mult &= op1 \times op2, \\ &= op1 \times \overline{b_{L-1}b_{L-2}\dots b_1b_0}, \\ &= op1 \times (b_{L-1}2^{L-1} + b_{L-2}2^{L-2} + \dots + b_1 2^1 + b_0 2^0), \\ &= op1b_{L-1}2^{L-1} + op1b_{L-2}2^{L-2} + \dots + op1b_1 2^1 + op1b_0 2^0. \end{aligned} \tag{1.1}$$

As $op1 \times b_i \times 2^i$ will not require much computation because b_i is a binary value and 2^i is a simple shift operation, we could say for practical purposes that this multiplication operation is equivalent to $L-1$ additions, i.e. $1\text{MPS} \simeq (L-1)\text{APS}$.

Based on this result we could define the total number of additions per second S_o as

$$S_o = (L-1)S_* + S_+ \tag{1.2}$$

to measure computation rate effect of both multiplications and additions, rather than measure them separately with S_* and S_+.

Depending on the application, the available resources and the filter parameters, the design could be implemented to reduce the computation rate (S_*, S_+ or S_o), the storage requirements (S_z) or both. For example, in an embedded system with a dedicated processor, the designer should prefer to reduce the computation rate than to reduce the storage space, but in very large-scale integration (VLSI) circuit the computation rate and storage reduction will be both critical. Because of that, it is also quantified:

- the estimated minimum frequency in a processor (f_{cpu}), assuming a single-core processor that executes an addition instruction in a clock cycle;
- the estimated number of adders in an FPGA running at 64 MHz (T^+_{FPGA});
- the estimated number of adders in a VLSI circuit running at 10 MHz (T^+_{lp}).

Those metrics will help the reader to have an idea of the required resources to implement the discussed and proposed methods in different platforms.

1.5 Assumptions

Filter specifications and array geometries will change depending on the beamformer application. Therefore, in order to compare the efficiency between the proposed and the state-of-the-art beamforming methods we make the following assumptions for the remainder of this thesis.

Decimation filter specification

The decimation filter specification shown in Table 1.1 is the base of all our decimation filter designs as it is considered enough for the most PDM-mic types and speech processing applications.

Parameter	Value
input sampling rate (f_i)	3072.0 kHz
output sampling rate (f_o)	16.0 kHz
passband frequency (F_p)	7.5 kHz
stopband frequency (F_s)	8.0 kHz
passband ripple (δ_p)	≤ 0.0116 (≤ 0.1 dB)
stopband ripple (δ_s)	≤ 0.0001 (≤ -80.0 dB)
decimation factor (R)	192
filter input length (L_{in})	1
filter output length (L_{out})	24
phase response	linear or almost linear

Table 1.1: Decimation filter specifications

Beamformer specification

As the delay from the array center to the mth microphone (Δ_m) in a symmetric array is constrained to

$$|\Delta_m| \leq \frac{|\bar{x}_{\max} - \bar{x}_c|}{c} \quad \text{for } m = 0, 1, \ldots, M-1\,, \tag{1.3}$$

where $\bar{x}_{\max}$ is the furthest sensor location, $\bar{x}_c$ is the array center reference, M is the number of microphones and c is the sound speed (typically 343.0 m/s); the maximum required delay ($\Delta_{\max}$) can be defined as

$$\Delta_{\max} = \frac{|\bar{x}_{\max} - \bar{x}_c|}{c}\,,$$

such that

$$|\Delta_m| \leq \Delta_{\max} \quad \text{for } m = 0, 1, \ldots, M-1\,. \tag{1.4}$$

Assume that we require a microphone array for hands-free applications that when placed 80 cm from the voice source would attain the same SNR than the SNR obtained by a single microphone placed to 2 cm from the same source, then, as the array gain (in dB) in an isotropic noise field is given by $20 \log M$ [5], the desired microphone array will require 40 microphones.

Also, as the minimum distance between microphones should be $D_{\min} \leq \frac{c}{2F_p}$ to avoid spatial aliasing; if the frequency range is limited to $F_p = 7.5\,\text{kHz}$, then the desired microphone array will require $D_{\min} \leq 2\,\text{cm}$.

Finally, as $M = 40$, if a 5×8 microphone array is assumed, then the $\Delta_{\max}$ can be calculated using (1.3) with resulting value shown in Table 1.2.

Parameter	**Value**
number of microphones (M)	40 (5×8)
minimum distance between microphones ($D_{\min}$)	22.0 mm
array dimensions	110.0 mm $\times$ 176.0 mm
maximum required delay ($\Delta_{\max}$)	314.47 µs
mth-filter channel gain (w_m)	1
frame length (for frequency domain implementations) (L_{frame})	4.0 ms

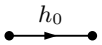
Table 1.2: Microphone array specifications

1.6 Notation

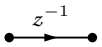
In the remainder of the thesis the following notation will be used with block diagrams:

- An arrow without any text in between the extremities is a simple connection.

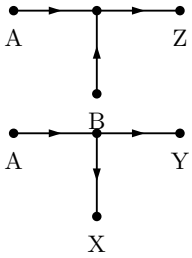
- An arrow with a number or variable in the middle, like h_0, represents a multiplication by that value.



- An arrow with z^{-1} in the middle represents a delay element.



- A node with two or more incoming arrows represents an adder (Z=A+B).
- A node with two or more outgoing arrows represents a distributor(X=A,Y=A).



1.7 Document organization

Chapter 2 presents a review about MEMS microphone technology, sigma-delta modulators and decimation filters.

Chapter 3 reviews the state of the art in efficient filter design and proposes a new algorithm to design efficient decimation filters with focus to application in PDM-mic array processing system (PAPS).

Chapter 4 reviews the conventional beamforming methods in discrete-time and frequency domains and proposes new approaches to do beamforming before decimation, i.e. on the PDM domain.

Chapter 5 reviews state of the art Samadi's filters and proposes new efficient beamformer implementation method based on it.

Chapter 6 summarizes the results and contributions and presents possible future work.

Chapter 2

Digital Microphone Technology

2.1 PDM microphones

According to [10], MEMS microphones, also known as PDM-mics, are currently the leading technology in the microphone market when it comes to number of sold units.

MEMS microphones can be realized exploiting different transduction principles such as piezoresistive, capacitive or optical detection. However, 80% of the produced MEMS microphones use the capacitive transduction principle, as this provides better sensitivity and low power consumption.

A capacitive MEMS microphone translates a variation in the sound pressure level to a variation of capacitance. As the capacitor is previously charged by a charge pump, the variation of capacitance causes a proportional variation in voltage level. This voltage variation is passed by a pre-amplifier to then be converted to digital signal by an ADC. A $\Sigma\Delta$ modulator ($\Sigma\Delta$M) is the preferred solution to implement the ADC in MEMS microphones because of its inherent linearity and low-power consumption [10]. The digital output from the $\Sigma\Delta$M associated with the MEMS microphone will be a PDM bitstream. It is expected that this PDM bitstream will be converted to a PCM signal using a decimation filter before further processing, as show in Figure 2.1.

The PDM bitstream is typically delivered at a sampling rate in the range of 1 MHz to 3 MHz, while the audio or baseband signal is supposed to be in the range of 20 Hz to 20 kHz. The $\Sigma\Delta$M order depends on the PDM-mic vendor, and they are generally second or higher order $\Sigma\Delta$M.

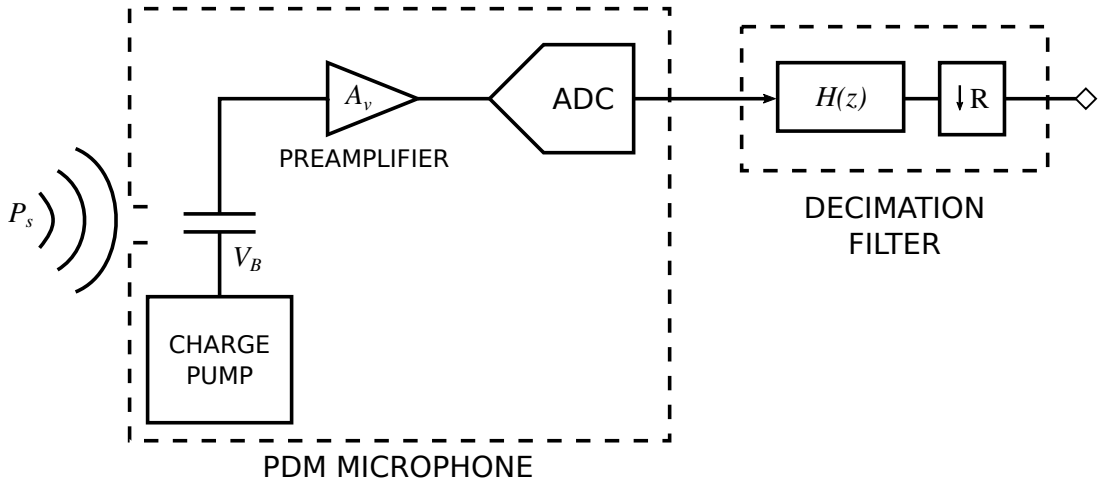


Figure 2.1: MEMS microphone block diagram

2.2 Sigma-delta ($\Sigma\Delta$) modulators

$\Sigma\Delta$M were first introduced in 1962 [11], but they only gained importance in the last decades, with the development of VLSI technologies; as these converters are based 90% on digital circuitry that can be integrated in the same die with other digital logic. This advantage alongside the use of digital techniques for audio processing and communications has contributed to the interest on this kind of cost effective high precision A/D converters.

The name *Sigma-Delta* modulator comes from putting an integrator (*Sigma*) alongside a comparator (*Delta*) in the analog-to-digital conversion circuit [11]. The integrator, as shown in the following paragraphs, shapes the quantization error to higher frequencies in the feedback system formed by the comparator.

In audio applications, the $\Sigma\Delta$M typically works at a sampling rate (f_i) of the order of a few megahertz while audio bandwidth is in the order of B_w=20 kHz, which results in a fairly large oversampling rate (OSR), where

$$\text{OSR} = \frac{f_i}{B_w}. \tag{2.1}$$

As shown in Figure 2.2a, in a $\Sigma\Delta$M, the analog signal $s(t)$ is first sampled at a rate f_i by a sample and hold circuit. The discrete signal $s[n]$ is then filtered and quantized to B bits by a feedback circuit. The feedback circuit shapes the quantization noise at higher frequencies while reducing noise in the baseband [12] as shown in Figure 2.3 for a 3 kHz tone applied to a 2^{nd}-order $\Sigma\Delta$M.

If it is assumed that the quantization error $e[n]$ is a wide-sense stationary signal, uncorrelated to $s[n]$, and it has a uniform distribution; then the B-bits nonlinear quantizer Q_B in Figure 2.2a could be modeled as a linear addition as shown in Figure 2.2b. So if the quantization error has the uniform probability density function as shown in Figure 2.4 for rounding or truncation cases and its variance is $\sigma_e^2 = \frac{\Delta^2}{12}$ where $\Delta = 2^{-B}$ is the smallest

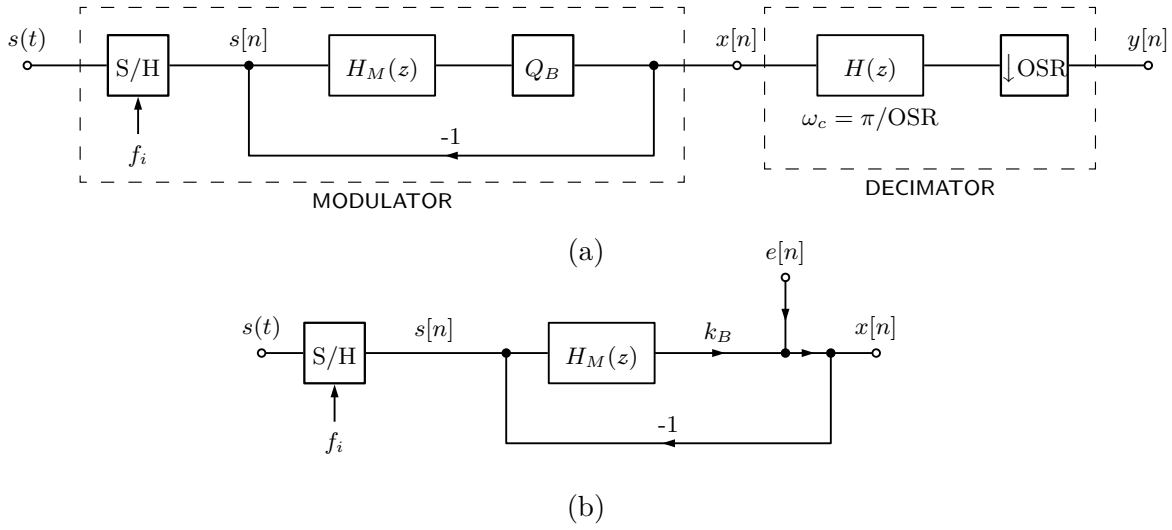


Figure 2.2: (a) ΣΔ modulator and decimator block diagram. (b) ΣΔ modulator linear model.

difference between two consecutive quantization levels; as its autocorrelation is

$$\phi_{ee}[n] = \sigma_e^2 \delta[n] + m_e \,,$$

where m_e is the error mean, its power density spectrum (defined as the Fourier transform of the autocorrelation function of stationary or wide-sense stationary signals) will be

$$P_{ee}(\omega) = F\{\phi_{ee}[n]\} = \sigma_e^2 = \frac{\Delta^2}{12} + 2\pi m_e \delta[n], \tag{2.2}$$

where $F\{\cdot\}$ is the Fourier transform [13].

The ΣΔM output could be expressed as

$$X(z) = \text{STF}(z)S(z) + \text{NTF}(z)E(z), \tag{2.3}$$

where $\text{STF}(z) = \frac{k_B H_M(z)}{1+k_B H_M(z)}$ is the signal transfer function and $\text{NTF}(z) = \frac{1}{1+k_B H_M(z)}$ is the noise transfer function.

Therefore, given a ΣΔM order L, if $H_M(z)$ is designed in such a way that

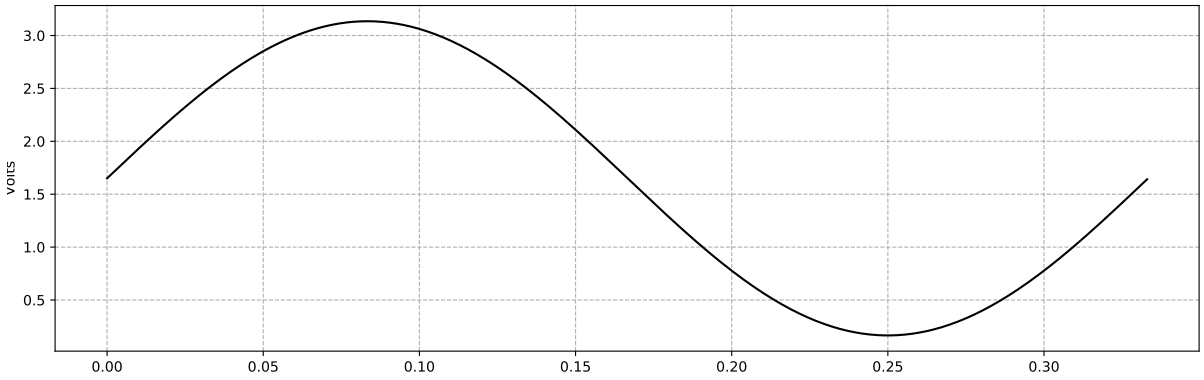
$$H_M(z) = \frac{1}{k_B(1-z^{-1})^L} - \frac{1}{k_B}$$

then

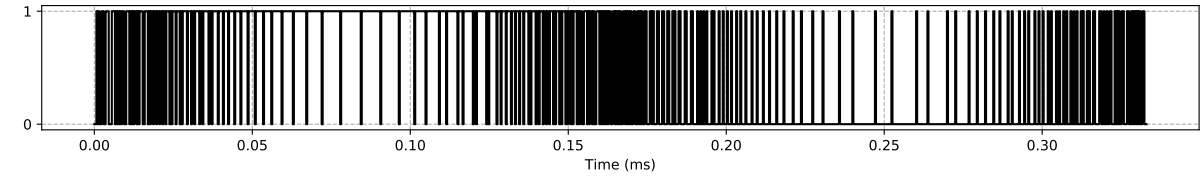
$$\text{NTF}(z) = (1-z^{-1})^L, \tag{2.4}$$

and as $z = e^{j\omega}$ the absolute value of $\text{NTF}(z)$ will be

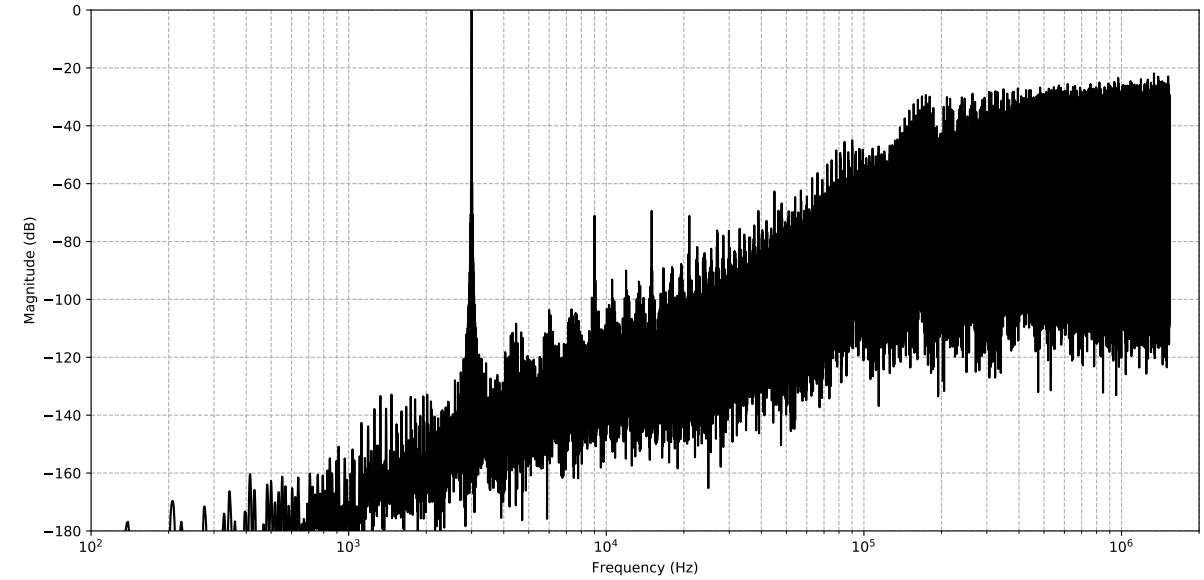
$$|\text{NTF}(e^{j\omega})| = |1-e^{-j\omega}|^L = \left|2\sin\left(\frac{\omega}{2}\right)\right|^L .$$



(a)



(b)

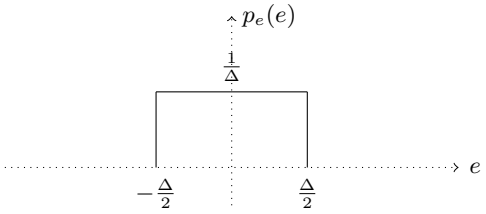


(c)

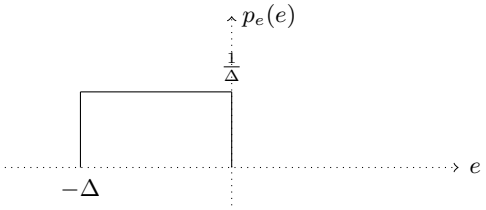
Figure 2.3: 2^{nd}-order $\Sigma\Delta$M (a) input, (b) output and (c) frequency spectrum

Also, in order to avoid aliasing and keep the signal in the B_w range, by (2.1), the decimation filter cutoff frequency is constrained to $\omega_c = \frac{2\pi B_w}{f_i} = \frac{\pi}{\text{OSR}}$. Therefore, if it is assumed that this decimation filter has an ideal low-pass impulse response

$$|H(e^{j\omega})| = \begin{cases} 1 & |\omega| \leq \omega_c, \\ 0 & |\omega| > \omega_c; \end{cases}$$



(a) Rounding



(b) Truncation

Figure 2.4: Quantization noise probability density function

by Parseval's theorem, the noise power in the decimator output $y[n]$ (Figure 2.2a) will be

$$P_e = \frac{1}{2\pi}\int_{\frac{-\pi}{\mathrm{OSR}}}^{\frac{\pi}{\mathrm{OSR}}} P_{ee}(\omega)|\mathrm{NTF}(e^{j\omega})|^2|H(e^{j\omega})|^2 d\omega = \frac{1}{2\pi}\int_{\frac{-\pi}{\mathrm{OSR}}}^{\frac{\pi}{\mathrm{OSR}}} \frac{\Delta^2}{12}\left[2\sin\left(\frac{\omega}{2}\right)\right]^{2L}|H(e^{j\omega})|^2 d\omega \tag{2.5}$$

but as $\left|2\sin\left(\frac{\omega}{2}\right)\right|^L \approx |\omega|^L$ when $\omega \ll 1$ (or $f \ll f_i$); the noise power can be approximated to:

$$P_e \approx \frac{1}{2\pi}\int_{\frac{-\pi}{\mathrm{OSR}}}^{\frac{\pi}{\mathrm{OSR}}} \frac{\Delta^2}{12}\omega^{2L} d\omega = \frac{\Delta^2}{12\pi}\int_0^{\frac{\pi}{\mathrm{OSR}}} \omega^{2L} d\omega = \frac{\Delta^2}{12\pi}\left(\frac{\omega^{2L+1}}{2L+1}\right)\Bigg|_{\omega=\frac{\pi}{\mathrm{OSR}}}. \tag{2.6}$$

Therefore, simplifying (2.6), the noise power will be:

$$P_e \approx \frac{\Delta^2}{12}\frac{\pi^{2L}}{(2L+1)\mathrm{OSR}^{2L+1}} \tag{2.7}$$

In the same way, if a sine tone is applied at $s(t)$ such that the amplitude of the sine tone at $x[n]$ is maximum, as $|H(e^{j\omega})| = 1$, the power output at the output $y[n]$ caused by the sine tone will be

$$P_s = \frac{\left(2^{B-1}\Delta\right)^2}{2}. \tag{2.8}$$

Finally, the ΣΔM dynamic range (DR) in the output $y[n]$ would be

$$\mathrm{DR} = \frac{P_s}{P_e} = \frac{2^{2B}3(2L+1)\mathrm{OSR}^{2L+1}}{2\pi^{2L}} \tag{2.9}$$

which shows how the DR increases exponentially with $2L + 1$ and justifies the usage of high order $\Sigma\Delta$M in MEMS microphones architectures. Note also that in the PDM-mic case, as its output $x[n]$ is a bitstream only, the number of bits in the quantizer Q_B will be $B = 1$.

2.3 Decimation filters

A decimation filter is a class of multirate filters [14] that decreases a signal sampling rate by an integer or fractional factor. Figure 2.5 shows a generic decimation filter block diagram, the input signal at f_i sampling rate passes through a low-pass filter (LPF) with impulse response $H(z)$ and then it is downsampled by an R factor to an output sampling rate $f_o = f_i/R$. In a PDM-mic case, usually $x[n]$ has 1-bit width only while $y[k]$ is a multi-bit output. This will be further explained later in this section.

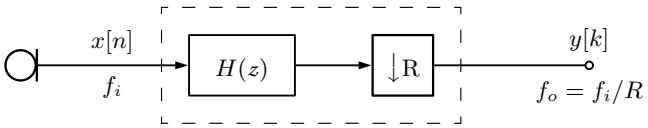


Figure 2.5: Single-stage decimation filter

For a given application, there are many design parameters to be taken into account for LPF design such as filter passband frequency F_p, stopband frequency F_s, passband ripple δ_p and stopband ripple δ_s[1], as exemplified in Figure 2.6. Those LPF design parameters are related as follows

$$U_p = \{f : f \in [0, F_p]\} \tag{2.10a}$$

$$U_s = \{f : f \in [F_s, f_i]\} \tag{2.10b}$$

$$\delta_p = \max(\big||H(e^{2\pi i \frac{f}{f_i}})| - 1\big|) \quad \forall f \in U_p, \tag{2.10c}$$

$$\delta_s = \max(|H(e^{2\pi i \frac{f}{f_i}})|) \quad \forall f \in U_s, \tag{2.10d}$$

where U_p and U_s are the passband and stopband frequency ranges respectively. Also, the angular passband and stopband frequencies can be expressed as

$$\omega_p = \frac{2\pi F_p}{f_i}, \tag{2.11a}$$

$$\omega_s = \frac{2\pi F_s}{f_i}, \tag{2.11b}$$

[1] δ_s and δ_p sometimes are expressed in dB such that $\delta_s(\text{dB}) = 20\log_{10}(\delta_s)$ and $\delta_p(\text{dB}) = 20\log_{10}(\delta_p + 1)$

and U_p and U_s intervals can be scaled to angular frequency domain as

$$V_p = \frac{2\pi U_p}{f_i}, \tag{2.12a}$$

$$V_s = \frac{2\pi U_s}{f_i}. \tag{2.12b}$$

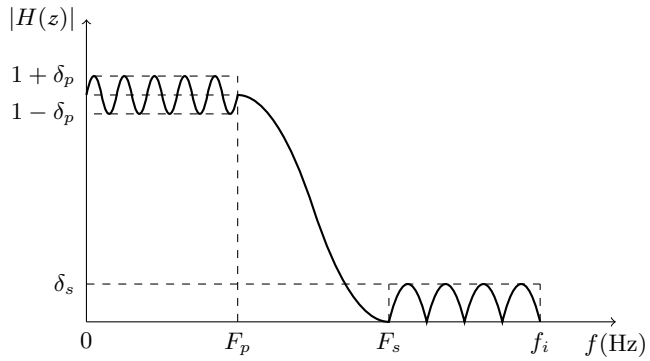


Figure 2.6: LPF design parameters

Baseband signal quality parameters such as linearity, SNR, total harmonic distortion (THD) and total harmonic distortion plus noise (THD+N) can be worsened at the filter output if the LPF is not properly designed [15]. Also the LPF structure should be carefully chosen to get a proper phase response; a Finite Impulse Response (FIR) structure, for example, can be used if linear phase is required; otherwise Infinite Impulse Response (IIR) filters are preferred as usually IIR filters are smaller than their equivalent FIR implementations. Also there are some applications that tolerates some degree of non-linearity; in this case quasi-linear filters, which are a mixture of FIR and IIR filters, can be used.

In this sense, for audio applications using PDM-mics, where decimation filters are required to get PCM audio signal from the oversampled PDM bitstream, the LPF is commonly implemented as an FIR filter, since in audio applications a linear phase response is required[2].

Also in the PDM-mic case, the input $x[n]$ is a 1-bit length oversampled signal that requires to be downsampled to standard audio sampling rates like 48 kHz or 16 kHz. During decimation, this 1-bit signal passes through several multiplications and additions, each operation increases sequentially the signal amplitude and consequently the number of bits of the signal. In the end, the filter output $y[k]$ is not a 1-bit length anymore, but a multi-bit signal.

[2]Linear phase is often required on audio applications due to the fact that such filters delay all frequencies by the same amount, thereby maximally preserving waveshape. Test results indicate that the perception of phase distortion is highly dependent on individual ability, and that it is easier to detect phase distortion by headphone listening rather than by loudspeaker listening [16].

2.3.1 Direct form FIR implementation

An FIR filter like the one discussed in the previous section can be implement in several ways. Figure 2.7a shows the direct form implementation structure for an FIR filter. In this implementation, $N-1$ delay elements are connected in series after the input, each delay element is multiplied by a filter coefficient h_n, and the multiplication result is added together sequentially by $N-1$ adders. Finally, the last adder result is downsampled by R. Figure 2.7b is the same direct form implementation but with multiplication done at the output rate, which will reduce the number of operations per second and consequently power consumption.

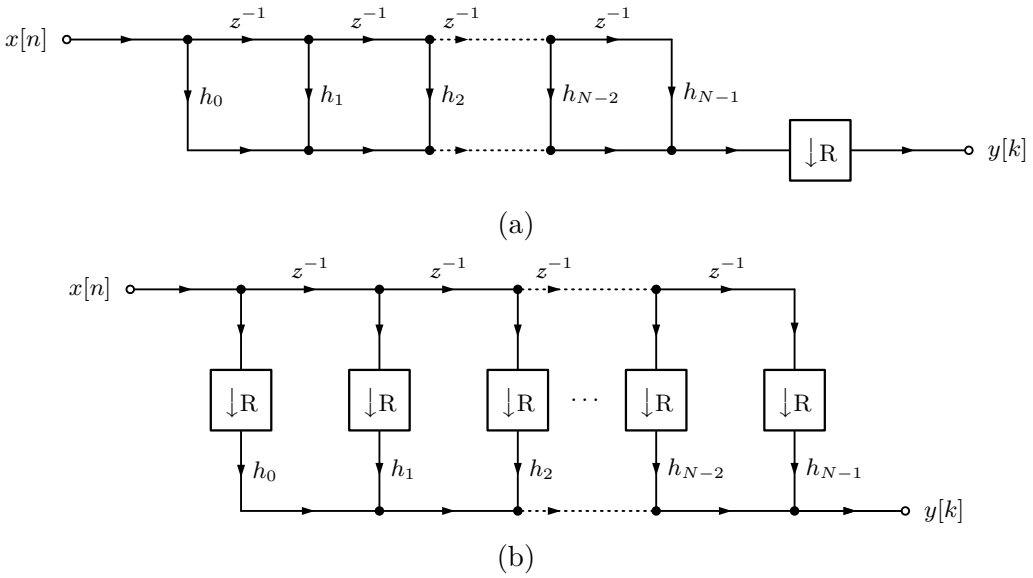


Figure 2.7: (a) Direct form and (b) efficient direct form implementations.

2.3.2 Polyphase FIR implementation

The polyphase decimation filter structure was first introduced on [17]. Here we assume that the decimation by a decimation factor R can be expressed as

$$y[k] = \sum_{s=-\infty}^{\infty} h[s]x[kR-s]. \tag{2.13}$$

Using the substitution $s = rR + \lambda$, for $\lambda \in \{0, \ldots, R-1\}$, the decimated signal can be expressed as

$$y[k] = \sum_{\lambda=0}^{R-1} \sum_{r=-\infty}^{\infty} h[rR+\lambda]x[(k-r)R-\lambda]. \tag{2.14}$$

To simplify (2.14), we define

$$h_\lambda[r] = h[rR + \lambda], \tag{2.15}$$

$$x_\lambda[r] = x[rR - \lambda], \tag{2.16}$$

such that

$$\begin{aligned} y[k] &= \sum_{\lambda=0}^{R-1} \sum_{r=-\infty}^{\infty} h_\lambda[r] x_\lambda[k-r] \\ &= \sum_{\lambda=0}^{R-1} y_\lambda[k] = \sum_{\lambda=0}^{R-1} h_\lambda[k] * x_\lambda[k]. \end{aligned} \tag{2.17}$$

Equation (2.17) is known as the polyphase representation of a decimation filter and it can be implemented as shown in Figure 2.8a, where all input downsamplers and delay elements can be replaced with a counterclockwise commutator at the filter input as shown in Figure 2.8b. Also it is easy to see from (2.16) that as the $h_\lambda[n]$ filters are interleaved versions of $h[n]$ by a factor of R, if $h_\lambda[n]$ filters are implemented as a transposed direct form implementation [18], memory elements are shared and the coefficients are sequenced at the sampling rate as shown in Figure 2.8c; then the number of delay elements are reduced by the same factor of R. This structure is known as the *efficient memory-saving* structure.

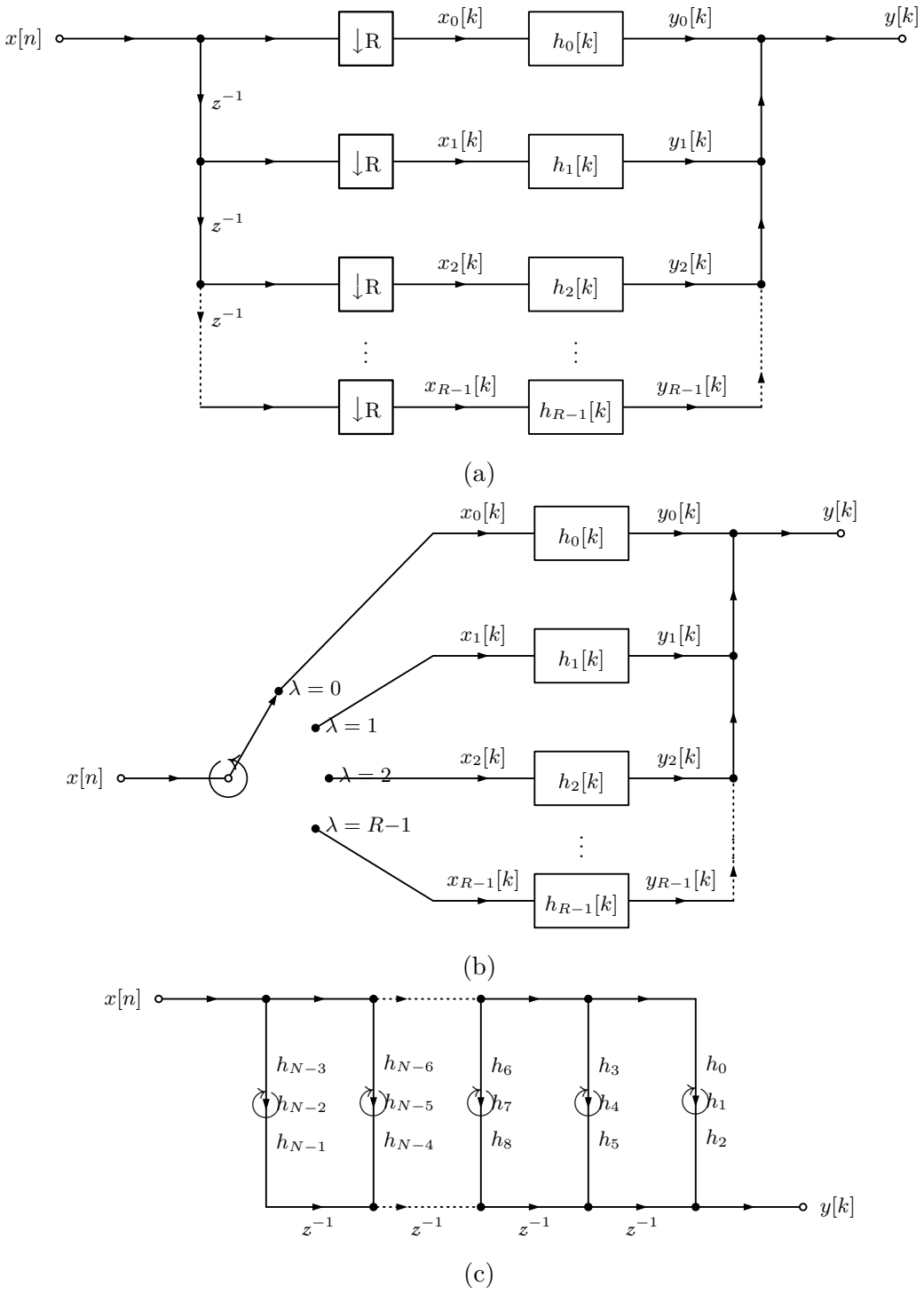


Figure 2.8: (a) Polyphase decimator, (b) polyphase decimator with input commuter and (c) memory-saving polyphase decimator ($R = 3$)

2.4 FFT implementation

As further discussed in [13], the FFT is an efficient way to calculate the Discrete Fourier transform (DFT) of a signal. If the frequency spectrum at all frequencies is required, the "butterfly" FFT implementation structure is the most efficient. As shown

in Figure 2.9, this structure is based in a sequence of repeated and interleaved sums and multiplications.

As this algorithm requires of an input buffer to save incoming data and a memory to store intermediary results during calculations, a D-points FFT will require $3DL$ bit or $4DL$ bit storage elements (S^z_{FFT}) depending if the input is real or complex respectively. Also, as the complexity of a D-points FFT algorithm in "butterfly" structure is $O(D) = D\log_2(D)$, it will also require $2D\log_2(D)f_o$ additions per second and $2D\log_2(D)f_o$ multiplications per second. These implementation resources are summarized in the following equation

$$S^+_{\mathrm{FFT}}(D, L, f_o) = 2D\log_2(D)f_o, \tag{2.18a}$$

$$S^*_{\mathrm{FFT}}(D, L, f_o) = 2D\log_2(D)f_o, \tag{2.18b}$$

$$S^z_{\mathrm{FFT}}(D, L, f_o) = \begin{cases} 3DL\,\text{bit} & \text{if input is real,} \\ 4DL\,\text{bit} & \text{if input is complex.} \end{cases} \tag{2.18c}$$

The number of storage elements is smaller when the input is real because just DL bit are used in the input, the others $2DL$ bit are used to store temporary results during calculation (imaginary and real parts). Also S^+_{FFT} and S^*_{FFT} are multiplied by 2 because operations are performed in both real and imaginary parts.

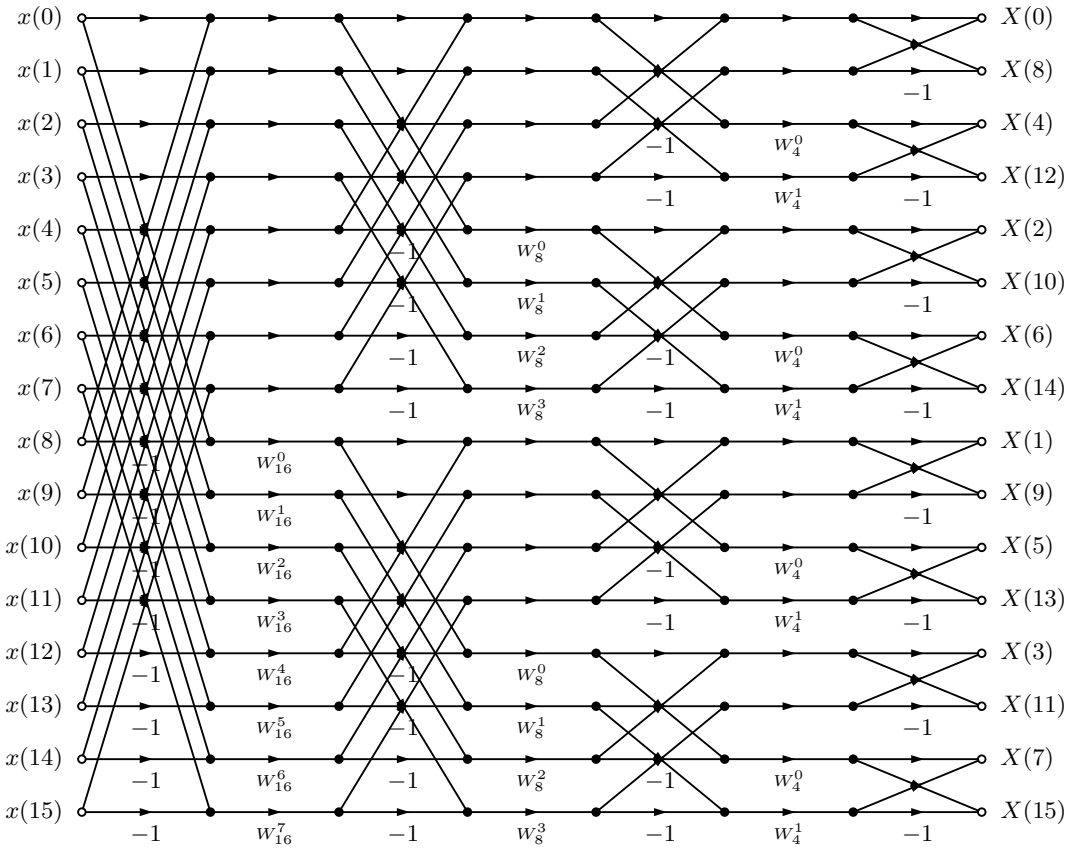


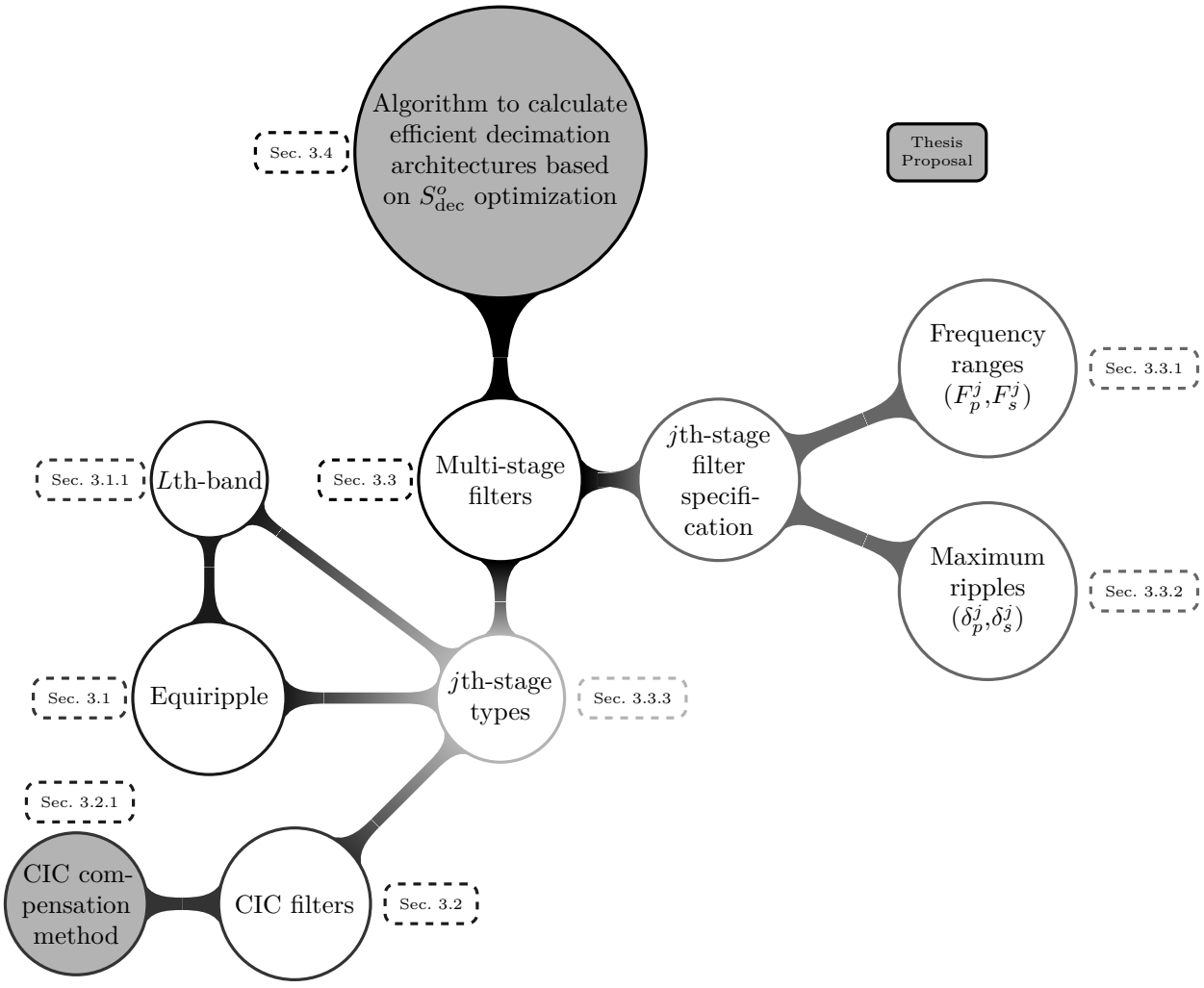
Figure 2.9: 16-points FFT implementation example with butterfly structure.

Chapter 3

Efficient multirate filter design

"Given a decimation filter specification, find an algorithm to determine the most efficient decimation filter architecture".

In order to fulfill the above objective already formulated in Section 1.2.1, this chapter proposes an algorithm to determine multirate filter architectures with minimal resource requirements. A bottom-up approach around the diagram below is followed to explain it.



So, at first, this chapter reviews the optimal equiripple FIR filter design method based on Parks-McClellan algorithm and its implementation structures. Three distinct FIR implementation structures are presented and the general expression to estimate the implementation resource requirements is derived. Furthermore, the Lth-band filter, a subclass of equiripple filters, is presented alongside the equations used to calculate them.

Next, CIC filters and CIC compensation filters are reviewed. The state of the art in CIC compensation filters is discussed and an additional compensation structure is proposed, as well as an algorithm to determine a CIC filter's order and its compensation filter given a filter specification.

It is important to remark that the window-based FIR filter design methods are not discussed as they are considered suboptimal [19]. Also, IIR filter discussion is avoided because for audio applications linear or almost linear phase response is considered a *de facto* standard. Linear phase filter are preferred because all the components of the input signal are equally shifted in time and consequently waveshape in the output is maximally preserved.

Once these relevant and efficient filter design methods are presented, general equations to determine the individual specifications of each stage in a multirate filter from a general specification are derived and the individual stage expression are related with the equiripple, Lth-band and CIC filters in a multirate filter.

Finally, an algorithm to determine multirate filter architectures with minimal decimation filter's total number of additions per second (S^o_{dec}) is formally proposed using previous results as basis and it is applied to obtain a set of efficient decimation filters with specifications as listed in Table 1.1. To conclude, the quantity of resources required to implement a beamformer using those efficient decimation filters is estimated.

3.1 Equiripple (optimal) FIR filters

The linear phase FIR filter design problem can be considered as a Chebyshev's approximation problem, such that the unique solution of this problem is optimal in the sense that the peak approximation error over the entire interval of approximation is minimized, as it is further discussed in [13, 20, 21]. Several approaches have been presented in the literature for solving this Chebyshev approximation problem. The solutions are based on either single-exchange linear programming solution [22], or multiple-exchange Remez algorithm solution developed by Parks and McClellan [23]. Parks-McClellan algorithm [24, 25] has become the standard method for FIR filter design because it is the most flexible and most computationally efficient. Parks-McClellan algorithm is already implemented in many commercial and open-source software as the Python's scientific library Scipy [26].

Given the LPF design parameters f_i, F_p, F_s, δ_s, δ_p and N there are empirical relationships found between them in [27] such that the minimum required FIR filter length N can be estimated from the relation

$$N = \frac{D_\infty(\delta_s, \delta_p)}{\Delta\bar{f}} - f(\delta_s, \delta_p)\Delta\bar{f} + 1, \tag{3.1}$$

where

$$\begin{aligned} D_\infty(\delta_s, \delta_p) =& (0.005309(\log_{10}\delta_p)^2 + 0.07114\log_{10}\delta_p + -0.4761)\log_{10}\delta_s \\ & - (0.00266(\log_{10}\delta_p)^2 + 0.5941\log_{10}\delta_p + 0.4278), \end{aligned} \tag{3.2}$$

$$f(\delta_s, \delta_p) = 11.01217 + 0.51244\log_{10}\left(\frac{\delta_s}{\delta_p}\right), \tag{3.3}$$

and for convenience, if transition bandwidth is defined as $\Delta f = F_s - F_p$, then normalized transition bandwidth $\Delta\bar{f}$ is defined as

$$\Delta\bar{f} = \frac{F_s - F_p}{f_i}. \tag{3.4}$$

The above relations are valid to within 1.3 percent relative error [27] in N if $\delta_s \leq 0.1$ and $\delta_p \leq 0.1$. Figure 3.1 shows how D_∞ decreases with δ_s (at a rate of 0.5 for every 20 dB) and δ_p (it decreases by 0.5 as δ_p increases from 0.05 dB to 0.2 dB). Also Figure 3.2, as both its axis are log, shows that N has an almost linear relationship with $\Delta\bar{f}$.

For applications as PDM-mic decimators, where $\Delta\bar{f} \ll 1$, the second and third terms in (3.1) are insignificant compared to the first term. Therefore, for convenience, this equation can be simplified to the form

$$N \simeq \frac{D_\infty(\delta_s, \delta_p)}{\Delta\bar{f}}. \tag{3.5}$$

The minimum required FIR filter length N for a single-stage decimation filter with the specifications listed in Table 1.1 and implemented with an equiripple (optimal) FIR LPF structure, as presented in Figure 2.7a, is estimated to be

$$N \simeq \frac{D_\infty(\delta_s, \delta_p)}{F_s - F_p} f_i \simeq 18976.$$

3.1.1 Lth-band equiripple filters

An Lth-band filter could require fewer multipliers than an equiripple one because, depending on the filter requirements, it could have many coefficients equal to zero. This special characteristic makes this filter very suitable for low-power implementations.

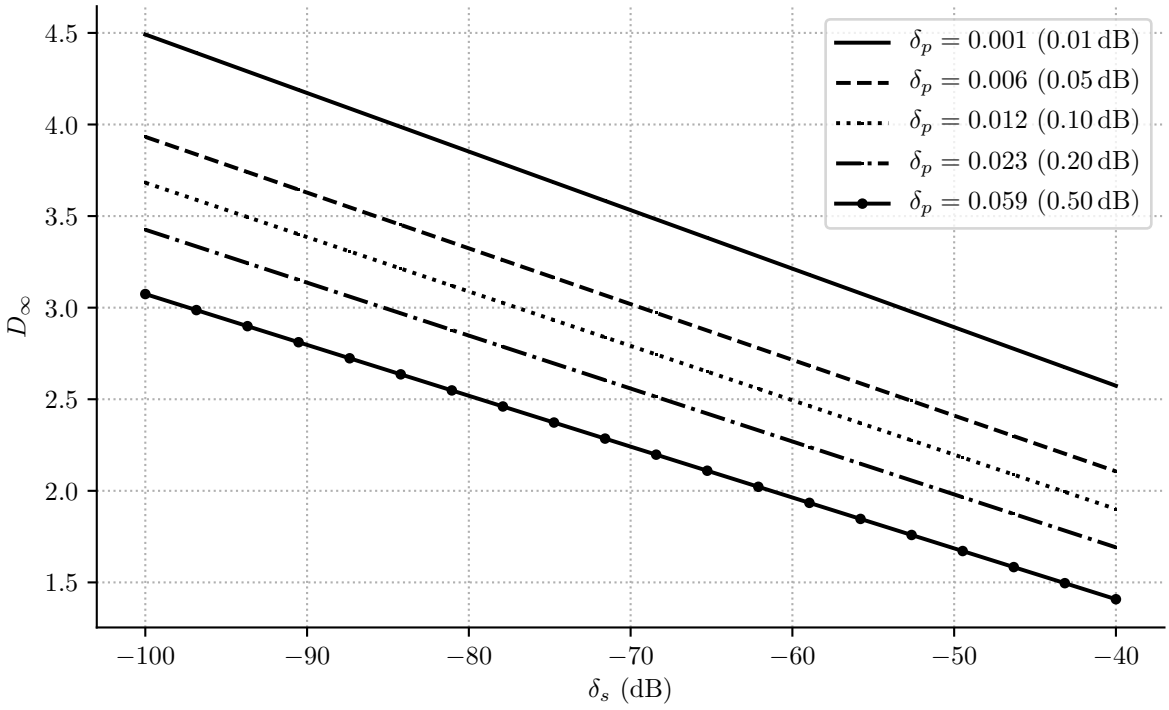


Figure 3.1: D_∞ for δ_s in $-100\,\mathrm{dB}$ to $-40\,\mathrm{dB}$ range.

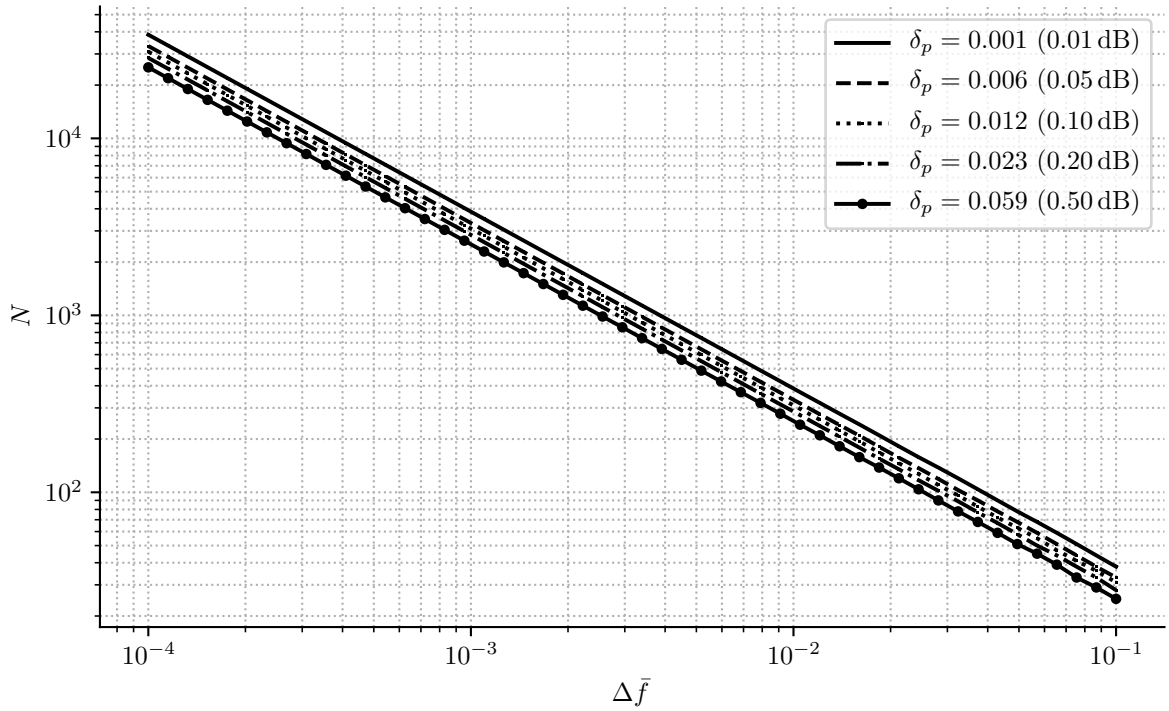


Figure 3.2: N for $\Delta\bar{f}$ in 1×10^{-4} to 1×10^{-1} range and $\delta_s = -80\,\mathrm{dB}$.

Its angular cutoff frequency is located at $\omega_c = \pi/L$ and its transition band is approximately symmetric around this frequency [14] such that, given a roll-off factor

$0 < \nu < 1$, its passband and stopband frequencies are related as

$$\omega_p = \frac{(1-\nu)\pi}{L} \tag{3.6}$$

$$\omega_s = \frac{(1+\nu)\pi}{L} \tag{3.7}$$

Also, if the Lth-band filter order is $N = 2K + 1$ where K is a positive integer, the impulse response coefficients satisfy the following conditions

$$h[K] = 1/L \tag{3.8}$$

$$h[K + rL] = 0 \quad \text{for } r \text{ in } 1,2,\ldots,\left\lfloor \frac{N}{L} \right\rfloor \tag{3.9}$$

where $\lfloor x \rfloor$, as know as the *floor* function, is defined as the integer part of x, such that the number of nonzero coefficients in the filter impulse response is

$$\begin{aligned} N_{nz} &= \left\lfloor N - \frac{N}{L} \right\rfloor \\ &= \left\lfloor \left(\frac{L-1}{L} \right) N \right\rfloor. \end{aligned} \tag{3.10}$$

Also, as shown in [28], the stopband ripple and passband ripple in an Lth-band filter are related by

$$\delta_p \leq (L-1)\delta_s. \tag{3.11}$$

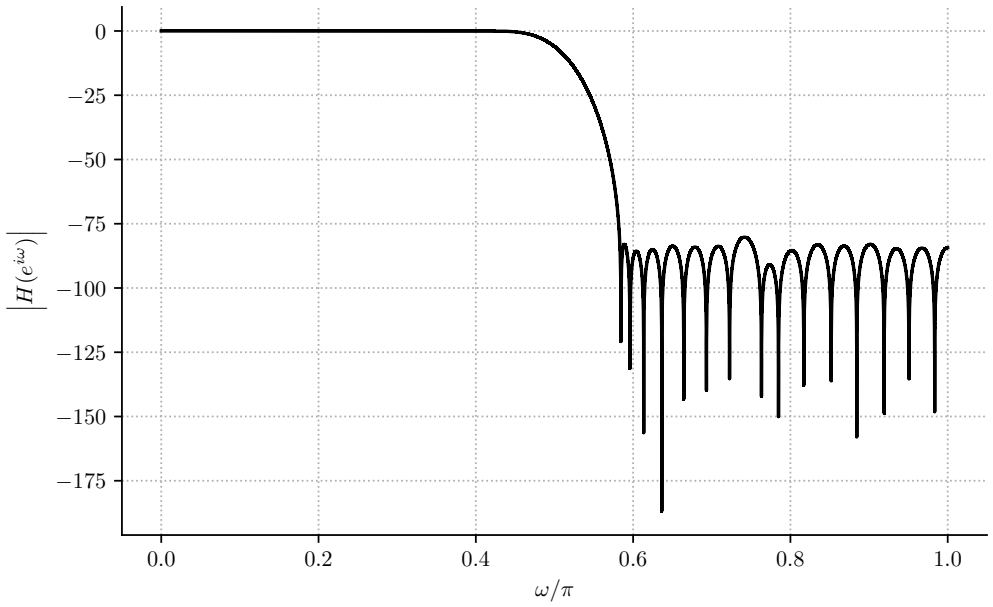
Figure 3.3 shows the frequency spectrum and impulse response of an Lth-band filter with $L = 2$ (half-band filter), it is shown that the half of the coefficients are zero.

3.1.2 FIR filter required resources

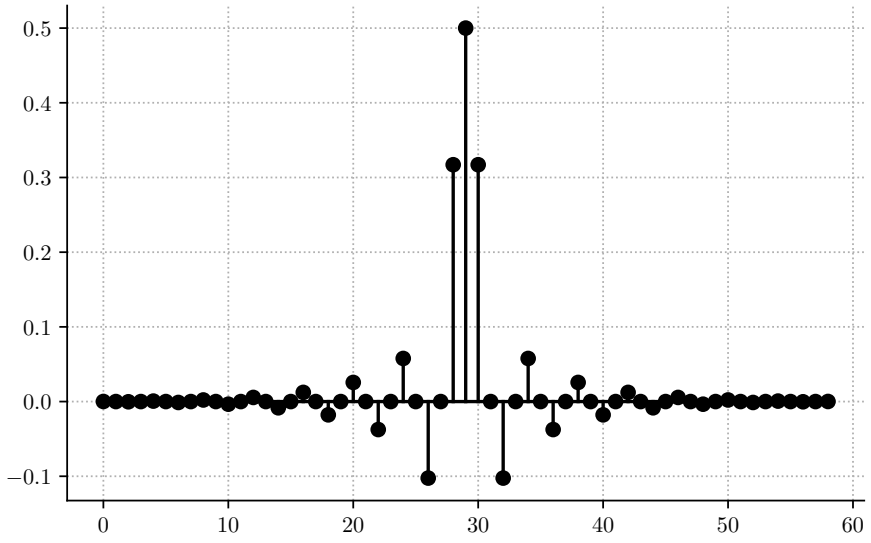
The resource requirements to implement FIR filters with direct form and polyphase structures are quantified in this subsection using the metrics proposed in Section 1.4.

Direct form implementation

In the direct form implementation of an FIR filter (shown in Figure 2.7a) the decimation filter's number of multiplications per second (S^*_{dec}) is given by the number of nonzero multipliers times the input sampling frequency, the decimation filter's number of additions per second (S^+_{dec}) is given by the number of adders times the input sampling frequency, and the decimation filter's storage requirement (S^z_{dec}) is given by the number of delay elements multiplied by the input's bit width. Therefore, the resource requirements



(a)



(b)

Figure 3.3: Lth-band filter (a) normalized frequency spectrum and (b) impulse response ($L = 2$).

for the direct form can be estimated as following:

$$S^{*}_{\text{dec}} = \begin{cases} \left\lceil \frac{\rho N}{2} \right\rceil f_i & \text{if } h[n] \text{ is symmetric,} \\ \rho N f_i & \text{otherwise,} \end{cases} \tag{3.12a}$$

$$S^{+}_{\text{dec}} = (\rho N - 1) f_i, \tag{3.12b}$$

$$S^{z}_{\text{dec}} = (N - 1) L_{\text{in}} \text{ bit}. \tag{3.12c}$$

The significance coefficient rate (ρ) is defined as

$$\rho = \frac{N_{\text{nz}}}{N}, \tag{3.13}$$

where N_{nz} is the number of nonzero coefficients in the filter impulse response and $\rho \simeq 1$ in most of the cases.

Efficient direct form implementation

Direct form implementation can be optimized to perform multiplications and additions at output sampling rate ($f_o = f_i/R$) as shown in Figure 2.7b such that the resource requirements are reduced to

$$S^*_{\text{dec}} = \begin{cases} \left\lceil \frac{\rho N}{2} \right\rceil \frac{f_i}{R} & \text{if } h[n] \text{ is symmetric,} \\ \rho N \frac{f_i}{R} & \text{otherwise,} \end{cases} \tag{3.14a}$$

$$S^+_{\text{dec}} = (\rho N - 1)\frac{f_i}{R}, \tag{3.14b}$$

$$S^z_{\text{dec}} = (N-1)L_{\text{in}} \text{ bit.} \tag{3.14c}$$

Memory-saving polyphase structure

In practice, the resource requirements to implement a decimation filter with a polyphase structure (shown in Figure 2.8b) are the same than these required for a direct form implementation given by (3.14). However, the resources required to implement an efficient memory-saving structure are given by

$$S^*_{\text{dec}} = \rho N \frac{f_i}{R}, \tag{3.15a}$$

$$S^+_{\text{dec}} = (\rho N - 1)\frac{f_i}{R}, \tag{3.15b}$$

$$S^z_{\text{dec}} = L_{\text{acc}} \left\lceil \frac{N}{R} \right\rceil \text{ bit,} \tag{3.15c}$$

where L_{acc} is the filter accumulator length.

Also, from comparing (3.14) and (3.15), it can be seen that memory-saving polyphase structure would be more efficient than efficient direct form implementation only if the LPF is not symmetric and L_{acc} and L_{in} meet the following relation

$$\frac{L_{\text{acc}}}{L_{\text{in}}} < R. \tag{3.16}$$

Table 3.1 shows the resource requirements to implement a single-stage decimation filter with specifications as listed in Table 1.1 using a direct form (*single_direct*), efficient direct form (*single_eff*) and memory-saving polyphase (*single_memsav*) implementations. It is shown that the *single_eff* implementation requires less total additions per second for the decimation filter but the same storage requirements, and the *single_memsav* implementation requires less storage requirements but more total additions per second. So, the *single_memsav* implementation will be preferred for low-power applications where only flip-flops are available, but for software implementations with large memory available, *single_eff* will be preferred as it requires less operations.

	S^{z}_{dec} (bit)	S^{*}_{dec} (MPS)	S^{+}_{dec} (APS)	S^{o}_{dec} (APS)	f_{cpu} (MHz)	T^{+}_{FPGA} (-)	T^{+}_{lp} (-)
single_direct	37950	29147136000	58291200000	87438336000	87438.34	1367	8744
single_eff	37950	151808000	303600000	455408000	455.41	8	46
single_memsav	396	303616000	303600000	607216000	607.22	10	61

Table 3.1: Single-stage decimation filter resource requirement comparison.

3.2 CIC filters

A Cascade Integrator-Comb (CIC) filter is an economical class of LPF introduced in [29] and widely used as the first stage of a multirate filter design. The main advantage of this class of filters is its multiplierless architecture and its high attenuation at stopband frequencies. Its main disadvantage is the non-flat frequency spectrum in the passband range, which needs to be compensated by other filter stages to ensure flatness in the overall filter passband range response.

The low-pass filter impulse response of a CIC filter decimated by R is given by

$$H(z) = \left(\frac{1}{R} \frac{1 - z^{-R}}{1 - z^{-1}} \right)^{K} \tag{3.17}$$

where K is the CIC filter order. The CIC decimation filter can be implemented as shown in Figure 3.4.

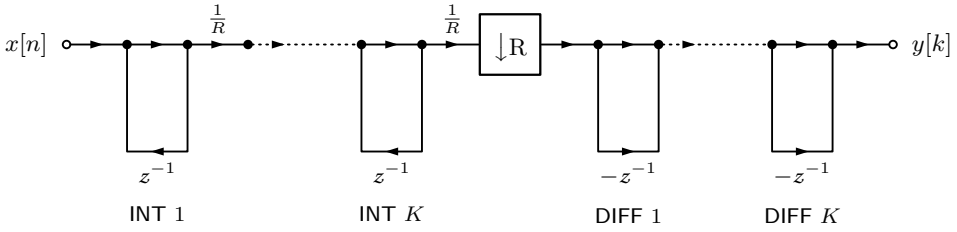


Figure 3.4: K-order CIC filter structure, a cascade of K integrators and K differentiators are required.

Figure 3.5 shows the magnitude of the K-stages CIC filter frequency spectrum which is given by

$$|H(e^{i\omega})| = \left| \frac{1}{R} \frac{\sin(\omega R/2)}{\sin(\omega/2)} \right|^K . \tag{3.18}$$

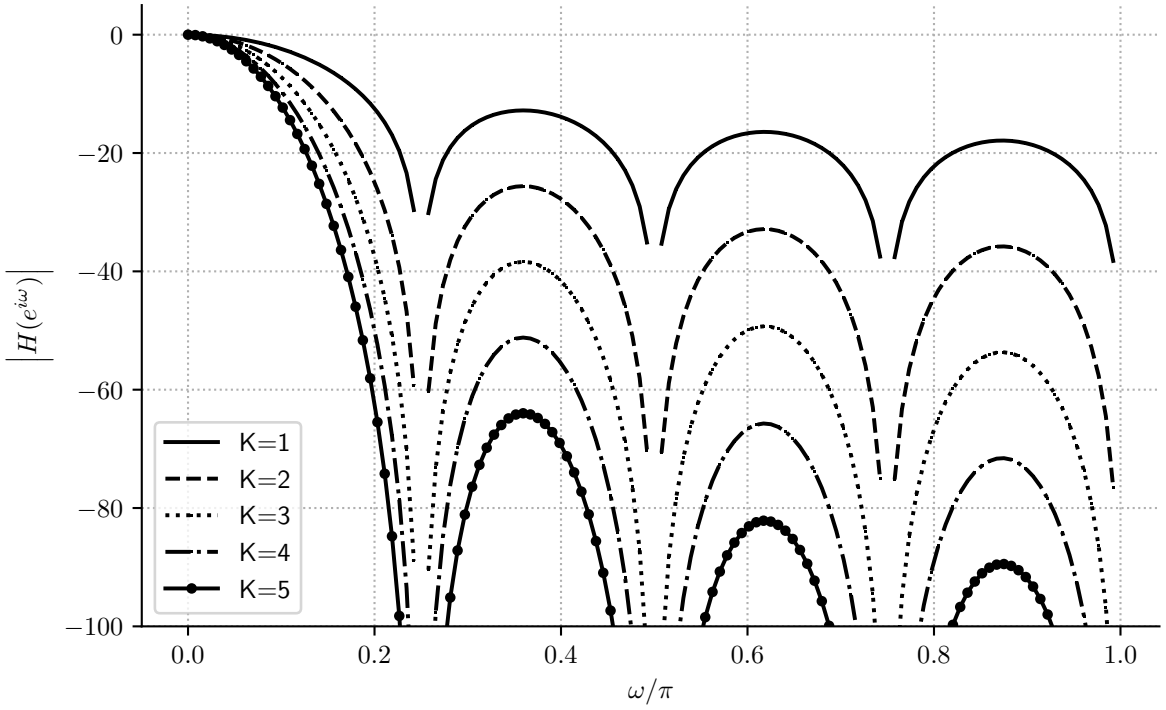


Figure 3.5: CIC filter frequency spectrum ($R = 8$).

3.2.1 CIC compensation

Due to the CIC filter passband droop usually it is required a compensation filter to keep the overall filter response under specification parameters. Those compensation filters are frequently designed as equiripple filters in the last stage of the multirate filter chain as shown in Figure 3.6a. As this approach require extra filter taps and multipliers, in the last decades there has been an effort in academia to design more efficient and multiplierless CIC compensation filter structures [1–3, 30–41]. In case of sparse FIR-based compensation filters, [41] proposes a structure that includes the compensation filter in the CIC structure using a time-varying multiplier that could improve performance and area. In some cases, proposed compensators are multiplierless implementations [3, 36, 37] but require to be redesigned for each value of decimation (R) or to have a maximum absolute deviation (passband ripple δ_p) larger than 0.2 dB. Other methods have a simple structure but require multipliers on their implementation [31, 34, 35]. Furthermore, recently published works [39, 40] propose multiplierless structures based on the particle swarm optimization (PSO) method to obtain compensator coefficients. These structures have

passband absolute deviation smaller than 0.1 dB and present a reconfigurable architecture for different values of K, but unfortunately they are not so flexible to changes of R and require complex structures for conversion to canonical signed digit (CSD) representation. All of these methods are reportedly outperformed by the methods proposed in [1] and [2], which have multiplierless structures, their filters depend only on the parameter K and their passband absolute deviation is smaller than 0.1 dB in either $c \leq 1/2$ (method [1]) or $c \leq 1/8$ (method [2]) ranges, where c is the normalized passband frequency in a CIC compensator and it is defined as

$$c = \frac{\omega_p}{\pi/R}. \tag{3.19}$$

As shown in Figure 3.6a, the method [1] proposes the use of two cascade filters after decimation whose frequency spectrums are

$$|G_1(e^{i\omega R})| = 1 + B_1 \sin^4(\omega R/2), \tag{3.20a}$$
$$|G_2(e^{i\omega R})| = 1 + B_2 \sin^2(\omega R/2), \tag{3.20b}$$

such that the proposed compensator filter has the following frequency spectrum:

$$|G_c(e^{i\omega R})| = (1 + B_1 \sin^4(\omega R/2))(1 + B_2 \sin^2(\omega R/2)). \tag{3.21}$$

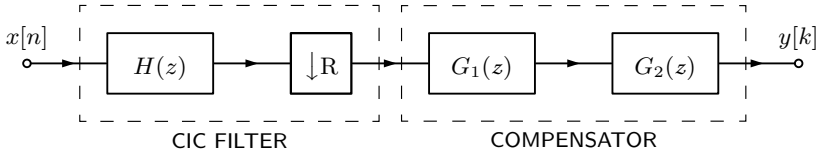
The method proposed in [2] can be regarded as a particular case of [1], where $|G_1(e^{i\omega R})| = 1$ and B_2 has other values. Figure 3.7 shows how the frequency spectrum of the CIC filter after compensation for different values of K is kept flat for $c \leq 1/2$ such that a decimator by 2 can be cascaded after the CIC compensation.

Methods [1, 2] have good compensation within $c \leq 1/2$ and $c \leq 1/8$ ranges, but sometimes it is required compensation at $c \leq 1/4$ range, which is between both ranges. In this case method [1] can be used but resources will be wasted unnecessarily. To overcome this limitation, based on [2], new B_2 coefficients can be heuristically calculated for a passband ripple $\delta_p \geq 0.1\,\text{dB}$[1] in the $c \leq 1/4$ range, as shown in Table 3.2. In Table 3.2 is also added the coefficients for method [3] that, as method [1], has good compensation within $c \leq 1/2$ but its passband ripple is $\delta_p \geq 0.4\,\text{dB}$. Figure 3.8 shows the passband ripple versus decimation factor and CIC filter order for proposal and [1–3] methods.

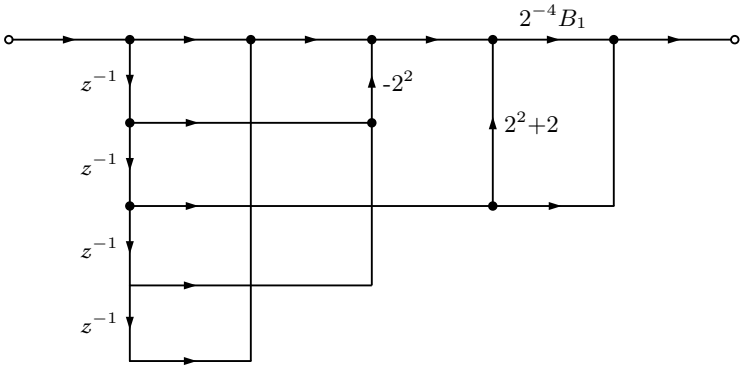
Finally, from (3.18) and (3.21), the overall CIC filter plus compensation impulse response can be expressed as

$$|H_{K,R,B_1,B_2}(e^{i\omega})| = \left|\frac{1}{R}\frac{\sin(\omega R/2)}{\sin(\omega/2)}\right|^K \left|(1 + B_1 \sin^4(\omega R/2))(1 + B_2 \sin^2(\omega R/2))\right|. \tag{3.22}$$

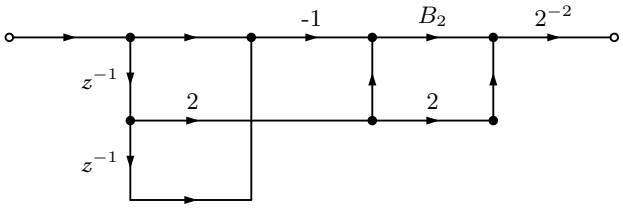
[1]It is important to remark that the expression "$\delta_p \geq 0.1\,\text{dB}$" means that the filter's passband ripple cannot be less than 0.1 dB. So, for instance, a filter that requires $\delta_p = 0.05\,\text{dB}$ cannot be designed with this method, however any filter that requires $\delta_p \geq 0.1\,\text{dB}$ is possible.



(a) CIC filter with compensation



(b) $G_1(z)$ filter



(c) $G_2(z)$ filter

Figure 3.6: Compensation filter proposed by [1].

where B_1 and B_2 can be selected from Table 3.2 such that $c = \frac{\omega_p R}{\pi}$ meet conditions thereof; otherwise B_1 and B_2 are zero and compensation should be done using other methods.

3.2.2 CIC filter design

From (3.22), the required CIC filter order (K) can be calculated as

$$K = \left\lceil \frac{\log_{10}(\delta_s) - \log_{10}\left(1 + B_1 \sin^4(\omega_s R/2)\right) - \log_{10}\left(1 + B_2 \sin^2(\omega_s R/2)\right)}{\log_{10}\left|\frac{1}{R}\frac{\sin(\omega_s R/2)}{\sin(\omega_s/2)}\right|} \right\rceil. \quad (3.23)$$

However, as B_1 and B_2 depend on the K value to be calculated, the iterative procedure shown in Algorithm 1 is required to calculate the optimum value of K given ω_p, δ_s, δ_p, R parameters and the stopband range V_s. In this procedure, first K is calculated for B_1 and B_2 equal to zero, and then K is adjusted progressively until it meets the desired condition.

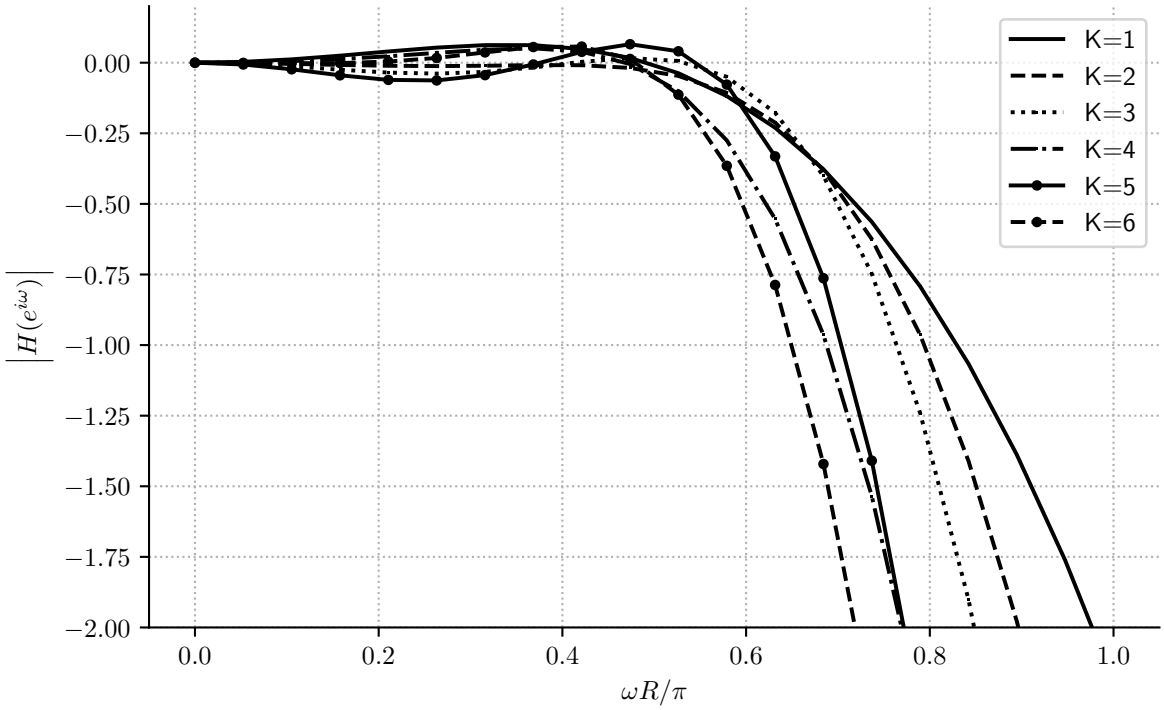


Figure 3.7: CIC compensation using method [1] ($c = 0.5, R = 16$).

3.2.3 CIC filter required resources

The resources required to implement a CIC filter are

$$S^{z}_{\text{dec}} \simeq 2KL_{\text{acc}} \text{ bit}, \tag{3.24a}$$

$$S^{*}_{\text{dec}} = 0, \tag{3.24b}$$

$$S^{+}_{\text{dec}} = Kf_o(1+R) + f_o N^{+}_{G_c}, \tag{3.24c}$$

where the number of adders for CIC compensator $N^{+}_{G_c}$ is selected from Table 3.2. Also, L_{acc} will be

$$L_{\text{acc}} = K \log_2(R|\max x[n]|) + 1, \tag{3.25}$$

where $x[n]$ is the filter input, and if the filter input is a bitstream

$$L_{\text{acc}} = K \log_2 R + 1. \tag{3.26}$$

Finally, as the passband requirements in Table 1.1 do not meet the required conditions from Table 3.2, it is not possible to implement a single-stage decimation filter with such specifications using a CIC filter followed by the multiplierless compensation architecture described in this section. However, this multiplierless compensator may be used in multi-stage decimation filter architectures as further discussed in following sections.

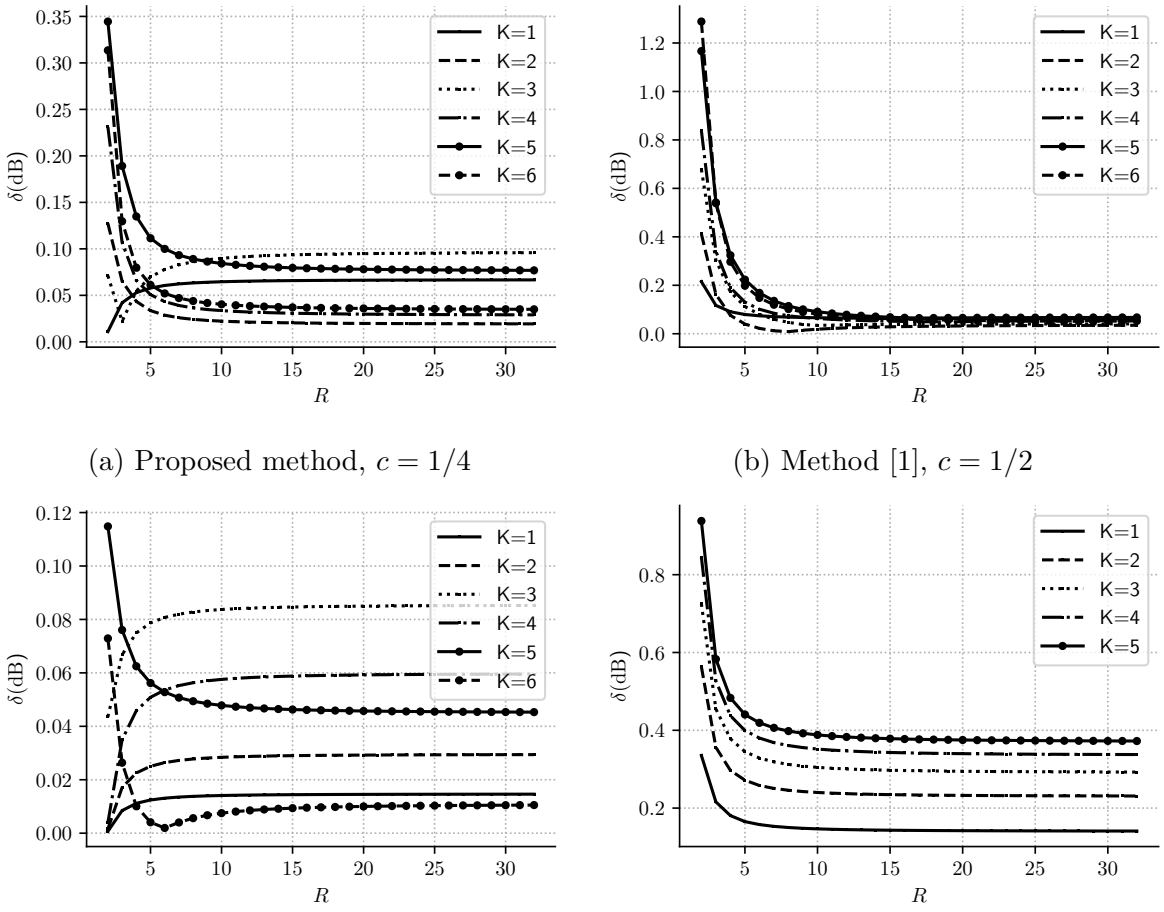
Method	Conditions	K	B_1	B_2	Adders ($N^+_{G_c}$)
proposal	$\delta_p \geq 0.1\,\text{dB}, c \leq 1/4, R > 5$	1	0	2^{-3}	3
		2	0	$2^{-2} + 2^{-3}$	4
		3	0	$2^{-2} + 2^{-3} + 2^{-4} + 2^{-5}$	6
		4	0	$2^{-1} + 2^{-2}$	4
		5	0	2^{0}	3
		6	0	$2^{0} + 2^{-3}$	4
[1]	$\delta_p \geq 0.1\,\text{dB}, c \leq 1/2, R > 10$	1	0	$2^{-2} - 2^{-5}$	4
		2	2^{-2}	$2^{-2} + 2^{-4}$	10
		3	2^{-1}	$2^{-1} - 2^{-4}$	10
		4	2^{-1}	$2^{-1} + 2^{-3} + 2^{-4}$	11
		5	1	$2^{0} - 2^{-2} - 2^{-5}$	11
		6	1	$2^{0} - 2^{-6}$	10
[2]	$\delta_p \geq 0.1\,\text{dB}, c \leq 1/8, R > 2$	2	0	2^{-2}	3
		3	0	2^{-2}	3
		4	0	2^{-1}	3
		5	0	2^{0}	3
		6	0	2^{0}	3
[3]	$\delta_p \geq 0.4\,\text{dB}, c \leq 1/2, R \geq 9$	1	0	2^{-2}	3
		2	0	2^{-1}	3
		3	0	$2^{-1} + 2^{-2}$	4
		4	0	2^{0}	3
		5	0	$2^{0} + 2^{-2}$	4

Table 3.2: CIC compensation filter's coefficients and number of adders. The listed methods are complementary to each other as the number of adders increases consistently with the passband ripple (δ_p), the normalized passband frequency in a CIC compensator (c) and the CIC filter order (K). So, any of them should be chosen properly depending on the particular filter requirements. It is also easy to see that the proposal, [1] and [2] methods have the same δ_p range but different passband range; and the methods [1, 3] have the same passband range but different δ_p, the latter one requiring more adders.

	Value	Unit
decimation filter's storage requirement (S^z_{dec})	1344	bit
decimation filter's number of multiplications per second (S^*_{dec})	0	MPS
decimation filter's number of additions per second (S^+_{dec})	6.4848e+07	APS
decimation filter's total number of additions per second (S^o_{dec})	6.4848e+07	APS
estimated minimum frequency in a processor (f_{cpu})	64.85	MHz
estimated number of adders in an FPGA running at 64 MHz (T^+_{FPGA})	2	-
estimated number of adders in a VLSI circuit running at 10 MHz (T^+_{lp})	7	-

Table 3.3: Single-stage decimation filter resource requirements implemented as a CIC filter, without compensation (K=21, B1=0, B2=0).

Table 3.3 shows the required resources to implement the referred filter specifications with a standalone CIC filter without compensation.



(a) Proposed method, $c = 1/4$

(b) Method [1], $c = 1/2$

(c) Method [2], $c = 1/8$

(d) Method [3], $c = 1/2$

Figure 3.8: CIC's passband ripple (δ_p) versus decimation factor (R) and CIC filter order (K). (a) Proposed method, whose coefficients are listed in Table 3.2, keeps the passband ripple $\delta_p \geq 0.1\,\text{dB}$ only for values of decimation factor $R > 5$ and the normalized passband frequency in a CIC compensator in $c < 1/4$ range. (b) Method [1] keeps $\delta_p \geq 0.1\,\text{dB}$ in the $c < 1/2$ range and $R > 10$. (c) Method [2] keeps $\delta_p \geq 0.1\,\text{dB}$ in the $c < 1/8$ range and $R > 2$. (d) Method [3], keeps $\delta_p \geq 0.4\,\text{dB}$ in $c < 1/2$ range and $R \geq 9$.

Algorithm 1 K calculation algorithm

procedure CICORDERCALC($\delta_s, \delta_p, R, \omega_p, V_s$)
    $B_1 \leftarrow 0$
    $B_2 \leftarrow 0$
    Calculate K ▷ Equation (3.24a)
    loop
        Calculate B_1, B_2 for $K = K$, $c = \frac{2F_pR}{f_{j-1}}$, $R = R$ ▷ Table 3.2
        ${\delta_s}' \leftarrow \max(|H_{K,R,B_1,B_2}(e^{i\omega})|) \quad \forall \omega \in V_s$ ▷ Ripple, Equation (3.22)
        if ${\delta_s}' \leq \delta_s$ **then**
            return K
        else
            $K \leftarrow K + 1$
        end if
    end loop
end procedure

3.3 Multi-stage filter design

Because of the high computation rate and storage requirements to implement a single-stage decimation filter, and consequently to implement a PAPS, if the decimation factor (R) can be factorized in J integer or fractional factors as

$$R = R_1 R_2 \dots R_J = \prod_{j=1}^{J} R_j$$

then the decimation filter can be divided in a cascade of J decimation filter stages as shown in Figure 3.9, where the jth-stage output sampling rate is

$$f_j = \frac{f_{j-1}}{R_j} \quad \forall j \in \{1, \dots, J\}, \tag{3.27}$$

and the jth-stage required filter output length is

$$L_j \geq \log_2(|\max y_j[n]|) + 1 \tag{3.28}$$

where $y_j[n]$ is the jth-stage filter output.

Besides the fact that, in general, the multi-stage implementation requires less implementation resources than its single-stage counterpart, the multi-stage filters are also more flexible to decimation rate changes and they allow to combine different filter structures, taking the best of any of them. For example, the high attenuation of a CIC filter could be combined with the flatness of a equiripple filter at different levels; or, as shown in Chapter 5, a MAXFLAT filter could be combined with equiripple or Lth-band filters to allow group delay configuration.

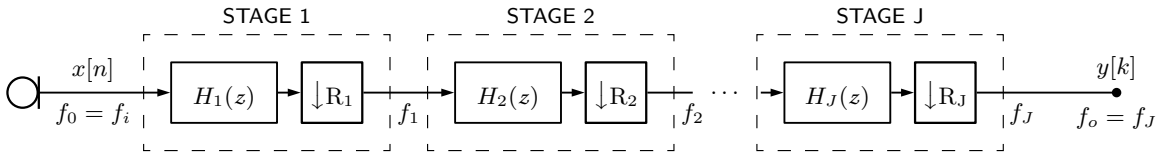


Figure 3.9: Multi-stage decimation filter

In this section are presented the passband and stopband frequency ranges and ripples of all internal individual stages of a multirate filter specified with parameters shown in Figure 2.6.

3.3.1 Passband and stopband frequency ranges

Equation (3.5) says that if the normalized transition bandwidth ($\Delta\bar{f}$) is maximized then the minimum required FIR filter length (N) is minimized proportionally. Therefore,

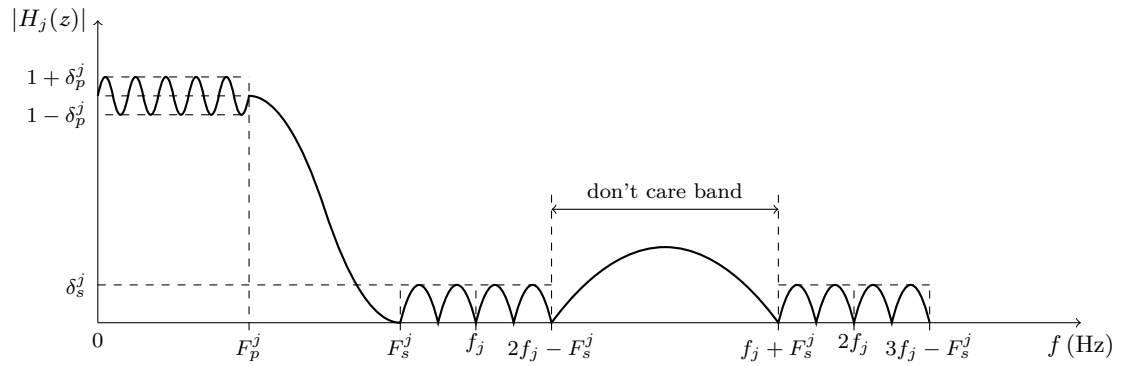


Figure 3.10: jth-stage filter prototype

the constraints presented in the following paragraphs are meant to maximize the allowed transition and don't care bands of each individual jth-stage in order to minimize its respective filter order. The individual jth-stage's filter order minimization will guarantee the overall minimization of resources of the combined J-stages decimation filter.

For all $j \in \{1, \ldots, J\}$ the jth-stage passband frequency range U_p^j is defined as

$$j \in \{1, \ldots, J\} \rightarrow U_p^j = \{f : f \in [0, F_p^j]\} \tag{3.29}$$

and the jth-stage stopband frequency range U_s^j is defined as

$$j \in \{1, \ldots, J-1\} \rightarrow U_s^j = S_{s1}^j \cup S_{s2}^j \cup S_{s3}^j \tag{3.30a}$$

$$j = J \rightarrow U_s^j = \left\{ f : f \in \left[F_s^j, \frac{f_{j-1}}{2} \right] \right\} \tag{3.30b}$$

such that

$$S_{s1}^j = \{f : f \in [(k-1)f_j + F_s^j, (k+1)f_j - F_s^j] \wedge k \in \{1, \ldots, \left\lfloor \frac{R_j}{2} - 1 \right\rfloor\}\}, \tag{3.30c}$$

$$S_{s2}^j = \{f : f \in [(k-1)f_j + F_s^j, (k+1)f_j - F_s^j] \wedge k = \left\lfloor \frac{R_j}{2} \right\rfloor \wedge R_j \text{ is odd}\}, \tag{3.30d}$$

$$S_{s3}^j = \{f : f \in [(k-1)f_j + F_s^j, kF_s^j] \wedge k = \left\lfloor \frac{R_j}{2} \right\rfloor \wedge R_j \text{ is even}\}, \tag{3.30e}$$

where F_p^j is the jth-stage passband frequency and F_s^j is the jth-stage stopband frequency. Figure 3.10 shows as the intervals in U_s^j are separated by f_j and they are interleaved by don't care bands. As it is further discussed in [42], those don't care bands reduce the filter complexity and consequently its length.

Also, if the jth-stage transition bandwidth is defined as

$$\Delta f_j = F_s^j - F_p^j, \tag{3.31}$$

it can be normalized as $\Delta\bar{f}_j$:

$$\Delta\bar{f}_j = \frac{\Delta f_j}{f_{j-1}}. \tag{3.32}$$

In order to keep the overall multirate filter response at stopband frequency (F_s) and passband frequency (F_p) specifications as shown in Figure 2.6, the jth-stage passband frequency value should be

$$\textit{if aliasing in transband is allowed:} \quad F_p^j = [F_p, F_s] \quad : j \in \{1, \ldots, J\}, \tag{3.33a}$$

$$\textit{otherwise:} \quad F_p^j = \begin{cases} [F_p, F_s] & : j \in \{1, \ldots, J-1\}, \\ F_p & : j = J, \end{cases} \tag{3.33b}$$

and the jth-stage stopband frequency value should be

$$\textit{if aliasing in transband is allowed:} \quad F_s^j = f_j - F_p \quad : j \in \{1, \ldots, J\}, \tag{3.34a}$$

$$\textit{otherwise:} \quad F_s^j = \begin{cases} f_j - F_s & : j \in \{1, \ldots, J-1\}, \\ F_s & : j = J. \end{cases} \tag{3.34b}$$

Finally, the jth-stage angular passband and stopband frequencies can be expressed as

$$\omega_p^j = \frac{2\pi F_p^j}{f_{j-1}}, \tag{3.35a}$$

$$\omega_s^j = \frac{2\pi F_s^j}{f_{j-1}}, \tag{3.35b}$$

and U_p^j and U_s^j intervals can be scaled to angular frequency domain as

$$V_p^j = \frac{2\pi U_p^j}{f_{j-1}}, \tag{3.36a}$$

$$V_s^j = \frac{2\pi U_s^j}{f_{j-1}}. \tag{3.36b}$$

3.3.2 Passband and stopband ripples

The equations (3.1) and (3.2) and Figure 3.1 show that the minimum required FIR filter length (N) decreases monotonically when the stopband ripple (δ_s) or passband ripple (δ_p) increase. So, in this section are derived the equations that maximize the individual δ_p (δ_p^j) and δ_s (δ_s^j) of each jth-stage in order to meet a given δ_p and δ_s for the overall filter.

If the jth-stage passband δ_p^j and stopband δ_s^j ripples are defined respectively $\forall j \in \{1, \ldots, J\}$ as

$$\delta_p^j = \max(\big||H_j(e^{2\pi i \frac{f}{f_{j-1}}})| - 1\big|) \quad \forall f \in U_p^j, \tag{3.37a}$$

$$\delta_s^j = \max(|H_j(e^{2\pi i \frac{f}{f_{j-1}}})|) \quad \forall f \in U_s^j, \tag{3.37b}$$

and the overall low-pass filter impulse response is

$$H(e^{2\pi i \frac{f}{f_i}}) = \prod_{j=1}^{J} H_j(e^{2\pi i \frac{f}{f_{j-1}}}) \quad \forall f \in U_p^j \cup U_s^j \tag{3.38}$$

where $H_j(e^{2\pi i \frac{f}{f_{j-1}}})$ is the jth-stage low-pass filter impulse response, the overall filter δ_p (for small δ_p) and δ_s can be expressed as

$$\delta_p = \sum_{j=1}^{J} \delta_p^j, \tag{3.39a}$$

$$\delta_s = \prod_{j=1}^{J} \delta_s^j. \tag{3.39b}$$

Even though only multi-stage decimation filters requirements are analyzed on this section, as further discussed in [20], the same filter requirements are valid for a multi-stage interpolation filter design, the only difference is the stage order that is inverted.

3.3.3 Multirate filter stages

In order to design a J-stages multirate filter with the same overall LPF specifications given by (2.10); all stages should meet conditions given by (3.29), (3.30), (3.33), (3.34), (3.37) and (3.39). So, depending on the application and the available resources; each stage should be designed, for instance, as an equiripple FIR, Lth-band or CIC filter to meet those individual stage requirements. This section points out some considerations to take into account when designing each jth stage with any of the mentioned structures.

Equiripple FIR filter stage

If the jth-stage LPF is designed as an equiripple FIR filter, from (3.5), (3.29) and (3.37) the jth-stage minimum required FIR filter length N_j can be estimated as

$$N_j \simeq \frac{D_\infty(\delta_s^j, \delta_p^j)}{\Delta \bar{f}_j}. \tag{3.40}$$

As it is desired to minimize N_j, assuming f_j and f_{j-1} fixed, F_p^j should be chosen to maximize $\Delta\bar{f}_j$. Then, $\Delta\bar{f}_j$ is maximum when $F_p^j = F_p$ if aliasing in transition band is allowed, otherwise $\Delta\bar{f}_j$ is maximum when $F_p^j = F_s$. Therefore, the optimal jth-stage minimum required filter length is

$$N_j \simeq \begin{cases} \dfrac{D_\infty(\delta_s^j, \delta_p^j)}{\Delta\bar{f}_j\big|_{F_p^j=F_p}} & : j \in \{1,\ldots,J\} \qquad \text{if aliasing in transband is allowed,} \\ \dfrac{D_\infty(\delta_s^j, \delta_p^j)}{\Delta\bar{f}_j\big|_{F_p^j=F_s}} & : j \in \{1,\ldots,J\} \qquad \text{otherwise.} \end{cases} \tag{3.41}$$

In most of the cases, the jth-stage minimum required FIR filter length has few zero-valued coefficients, such that it is reasonable to approximate the jth-stage significance coefficient rate as $\rho_j \simeq 1$.

Finally, once N_j and ρ_j are known, the required resources to implement the jth-stage using *single_direct*, *single_eff* or *single_memsav* structures can be estimated by (3.12), (3.14) or (3.15) respectively where $N = N_j$, $\rho = \rho_j$, $f_i = f_{j-1}$, $R = R_j$, $L_{\text{in}} = L_{j-1}$ and $L_{\text{acc}} = L_j$.

Lth-band FIR filter stage

In a J-stages multirate filter, the jth-stage LPF can be designed as a jth-band filter of order Rj if the following conditions are met:

- The filter is symmetric around π/R_j, such that if aliasing in transition band is allowed, the jth-stage passband frequency is $F_p^j = F_p$ for $j \in \{1,\ldots,J\}$; otherwise, if aliasing is not allowed, $F_p^j = F_p$ for $j \in \{1,\ldots,J\}$ i.e. the last stage cannot be a R_Jth band filter.

- The jth-stage design stopband and passband ripples meet the relation

$$\delta_j{}^{\text{band}} \leq \min(\delta_p^j, \delta_s^j). \tag{3.42}$$

Once both conditions are met, the optimal jth-stage minimum required filter length can be estimated by

$$N_j \simeq \begin{cases} \dfrac{D_\infty(\delta_j{}^{\text{band}}, \delta_j{}^{\text{band}})}{\Delta\bar{f}_j\big|_{F_p^j=F_p}} & : j \in \{1,\ldots,J\} \qquad \text{if aliasing in transband is allowed,} \\ \dfrac{D_\infty(\delta_j{}^{\text{band}}, \delta_j{}^{\text{band}})}{\Delta\bar{f}_j\big|_{F_p^j=F_s}} & : j \in \{1,\ldots,J-1\} \qquad \text{otherwise.} \end{cases} \tag{3.43}$$

Also from (3.10), the jth-stage significance coefficient rate is

$$\rho_j \simeq \frac{R_j - 1}{R_j} \tag{3.44}$$

Finally, once N_j and ρ_j are known, the required resources to implement the jth-stage using *single_direct*, *single_eff* or *single_memsav* structures can be estimated by (3.12), (3.14) or (3.15) respectively where $N = N_j$, $\rho = \rho_j$, $f_i = f_{j-1}$, $R = R_j$, $L_{\text{in}} = L_{j-1}$ and $L_{\text{acc}} = L_j$.

CIC filter stage

Given δ_s^j, R_j, δ_p^j, ω_p^j and V_s^j, the jth-stage CIC filter order (K_j) can be calculated using Algorithm 1 as shown in Figure 3.11.

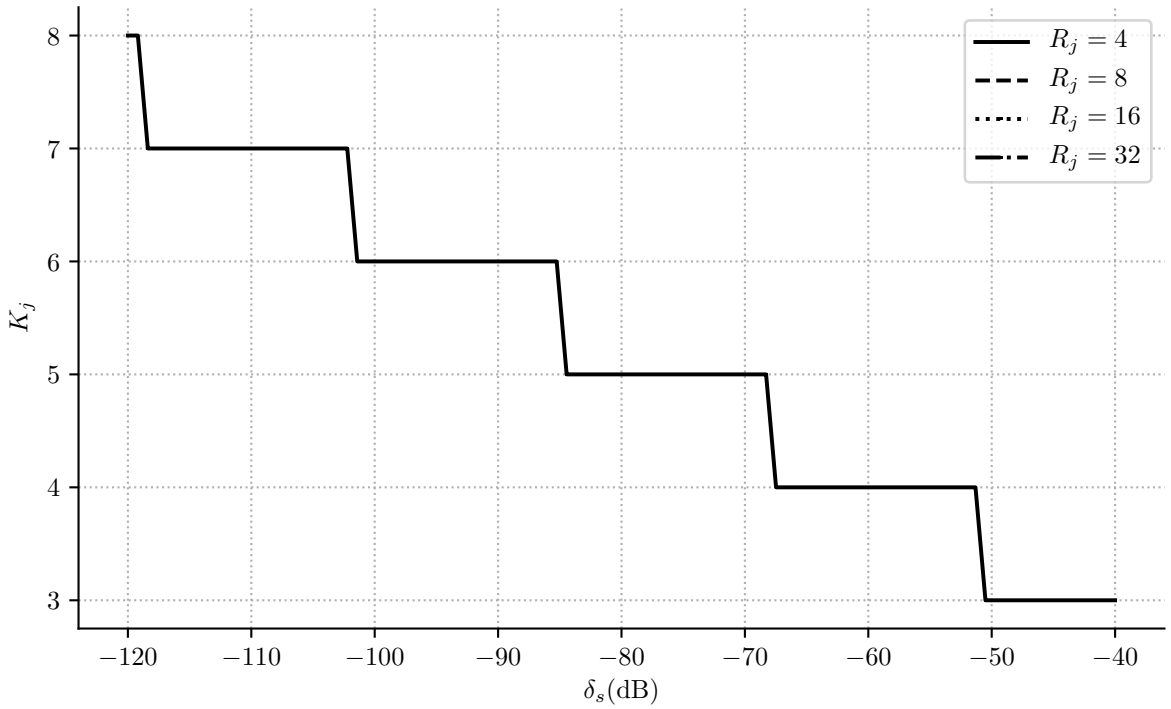


Figure 3.11: K_j for various values of R_j, assumming a CIC filter located at the first stage of a multirate filter with specifications as listed in Table 1.1 but δ_s varying.

Finally, the resources required to implement a jth-stage CIC filter can be estimated by (3.24) for $L_{\text{acc}} = L_j$, $K = K_j$, $f_o = f_j$ and $R = R_j$.

3.4 Proposal: Multirate filter design method based on S^o_{dec} optimization

Given a desired filter specification as shown in Table 1.1, the goal is to find an efficient multirate filter with J stages which requires the minimum implementation resources. As it is explained in Section 1.4, the suitable metric parameter that gives us implementation complexity information is the decimation filter's total number of additions per second (S^o_{dec}), so in this section is proposed an algorithm whose objective is to find out a J-stages multirate filter configuration with minimum S^o_{dec}. For other applications, where memory is critical for example, the filter design could be optimized around S^z_{dec} instead.

Then, the three parameters to be calculated to achieve our goal are:

- number of stages (J).
- the optimum $\{R_j\}$ combination for all $j \in \{1, \ldots, J\}$, and
- the jth-stage's type,

where the stage type could be any of the filter types studied on this thesis: equiripple (*equir*), Lth-band (*lthband*), self-compensated CIC (*cicfil*), standalone CIC (*ciconly*) or compensation filter (*comp*). The algorithm could be extended to accommodate any other filter implementation structure. Unfortunately, because of the numerous parameters and equations that need to be taken into account for each filter stage design, it is not straighforward to find a closed form to calculate all parameters mentioned above. So, Algorithm 2 is used to iterate over all possible filter configurations and get the filter with minimum S^o_{dec}.

Algorithm 2 Multirate filter design method based on S^o_{dec} optimization

```
1: procedure OptimumFilter(f_i, f_o, F_p, F_s, δ_p, δ_s, f_o, R, L_in, L_acc, aliasing)
2:     RFactorsList ← AllFactorsList(R)
3:     FilterList ← []
4:     for RFactors in RFactorsList do
5:         J ← length(RFactors)
6:         TypeStagesList ← AllCombinations(['cicfil','lthband','comp','equir','ciconly'],J)
7:         for TypeStages in TypeStagesList do
8:             Filter ← CalcMultiFilter(J, RFactors, TypeStages, f_i, f_o, F_p, F_s, δ_p, δ_s, f_o, L_in, L_acc, aliasing)
9:             Append(FilterList, Filter)
10:            S^o_dec ← CalcSo(Filter)
11:            Append(S^o_decList, S^o_dec)
12:        end for
13:    end for
14:    S^o_decMin ← min(S^o_decList)
15:    index ← Index(S^o_decList, S^o_decMin)
16:    return FilterList[index]
17: end procedure
```

Finally, the methods used in Algorithm 2 are:

- AllFactorsList(x) calculates a list of all possible factors of x, in reverse order, such that for example AllFactorsList(64) = $[\{32, 2\}, \{16, 4\}, \{8, 4, 2\}, \{4, 2, 2\}, \dots]$.
- AllCombinations($List, x$) calculates a list of x elements with all the possible combinations of *List*'s elements, such that for example AllCombinations($[a, b, c], 2$) = $[\{a, a\}, \{a, b\}, \{b, a\}, \{a, c\}, \dots]$
- CalcMultiFilter($J, Rfactors, TypeStages, \dots$) calculates a J-stages multirate filter with $R_j = Rfactors[j]$, jth-stage type equal to $TypeStages[j]$, and so on, for all $j \in \{1, \dots, J\}$.
- CalcSo($Filter$) calculates the S^o_{dec} from the specified filter $Filter$.
- Append($List, x$) appends the element x to the end of the list $List$.
- Index($List, x$) returns the index of x in $List$, provided that $x \in List$.

3.5 Results

Algorithm 2 was used to find the optimum filter structure for Table 1.1 specification that minimizes S^o_{dec}. The algorithm iterated over all possible filter configuration and the 10 best results regarding S^o_{dec} optmization parameter are shown in Table 3.4 and Table 3.5 sorted by the number of S^o_{dec} resources.

	R_j	Stage types
multi_0	96, 2	*lthband, equir*
multi_1	48, 2, 2	*lthband, lthband, equir*
multi_2	6, 2, 2, 2, 2, 2	*ciconly, cicfil, equir, equir, equir, equir*
multi_3	6, 2, 2, 2, 2, 2	*ciconly, ciconly, equir, equir, equir, equir*
multi_4	4, 3, 2, 2, 2, 2	*cicfil, cicfil, equir, lthband, lthband, equir*
multi_5	24, 2, 2, 2	*lthband, lthband, lthband, equir*
multi_6	48, 2, 2	*lthband, equir, equir*
multi_7	16, 3, 2, 2	*lthband, equir, lthband, equir*
multi_8	24, 2, 2, 2	*lthband, equir, lthband, equir*
multi_9	24, 4, 2	*lthband, lthband, equir*

Table 3.4: Multi-stage decimation filters found by the optimization algorithm.

The result shows that the filter *multi_0* based on Lth-band filter is the most efficient configuration for this filter specification. The *multi_0* filter overall frequency spectrum is presented in Figure 3.12 and the frequency spectrum of each individual filter stage is shown in Figure 3.13.

It is also interesting to see that the third and fourth most efficient filters, *multi_2* and *multi_3* respectively, are novel architectures based on interleaved CIC and equiripple

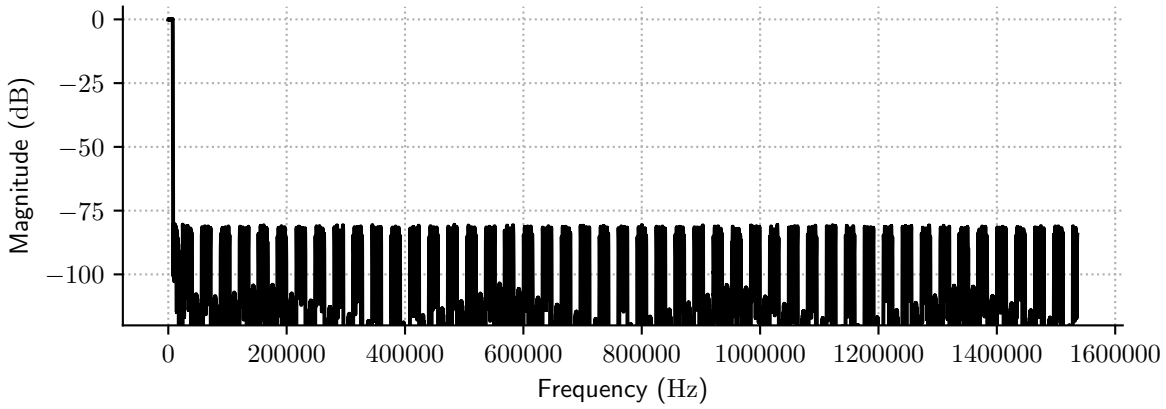
	S^z_{dec} (bit)	S^*_{dec} (MPS)	S^+_{dec} (APS)	S^o_{dec} (APS)	f_{cpu} (MHz)	T^+_{FPGA} (-)	T^+_{lp} (-)
multi_0	4963	30320000	30272000	104240000	104.24	2	11
multi_1	5669	19248000	19136000	110496000	110.50	2	12
multi_2	7666	3984000	19616000	110912000	110.91	2	12
multi_3	7666	3984000	19616000	110912000	110.91	2	12
multi_4	7177	3488000	27824000	113584000	113.58	2	12
multi_5	5762	15408000	15168000	117344000	117.34	2	12
multi_6	6371	19840000	19728000	126224000	126.22	2	13
multi_7	5794	9712000	14400000	126304000	126.30	2	13
multi_8	6042	15776000	15536000	126960000	126.96	2	13
multi_9	5740	15984000	15808000	128320000	128.32	3	13

Table 3.5: Comparison of multi-stage decimation filters found by the optimization algorithm.

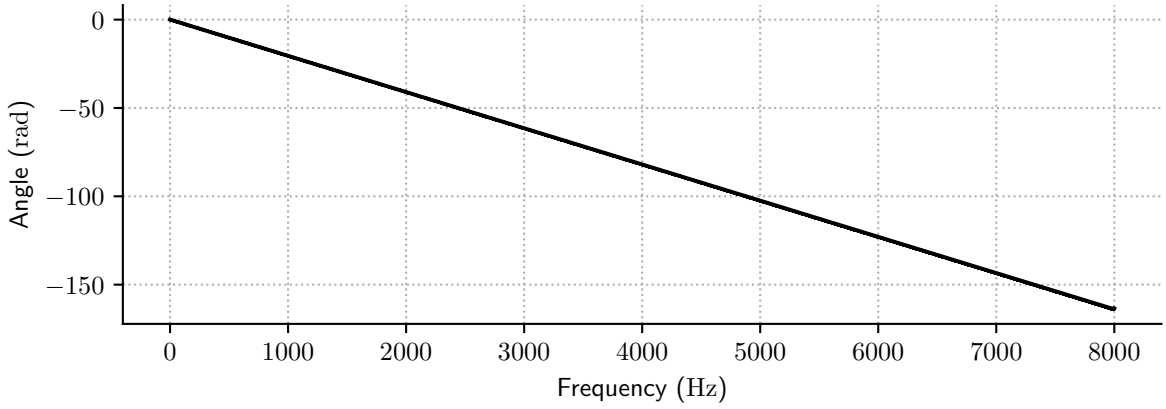
filters. For some application these architectures based on CIC filters could be more convenient because of its flexibility to change the decimation rate without changing the filter coefficients or its structure.

Finally, in a PAPS implementation case, as shown in Figure 3.14 , as the filters architectures found by Algorithm 2 are parallelized, one per microphone input, the required resources would be multiplied by the number of microphones M. Table 3.6 shows the resources required to implement a PAPS for 40 microphones.

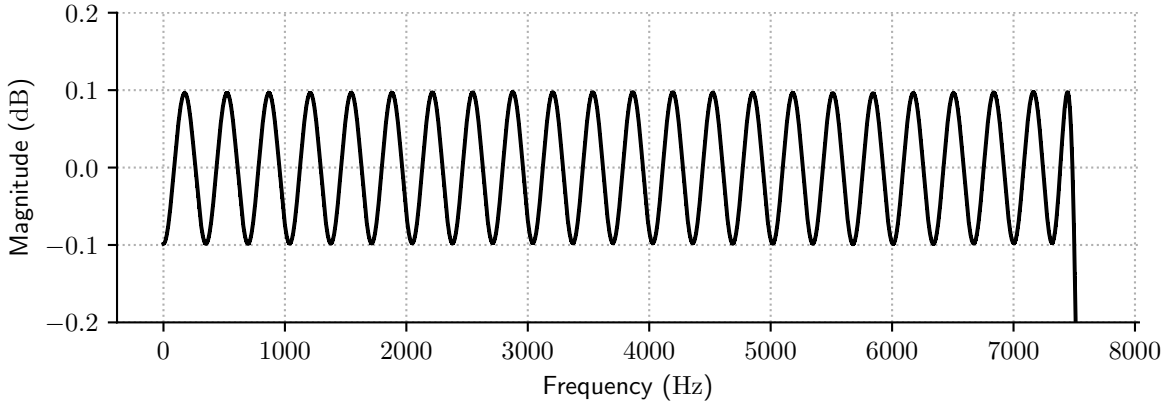
The S^z_{dec}, S^+_{dec}, S^*_{dec} and S^o_{dec} columns in Table 3.6 are the same than Table 3.5's ones multiplied by 40 because of each independent decimation channel. As the implementantion in a single-core processor does not allow resource sharing, the f_{cpu} is also proportional by 40 to the respective value in Table 3.5, resulting in prohibitive resource requirements of more than 4 GHz processor frequency. But as implementation in Field Programmable Gate Array (FPGA) or VLSI allows some degree of resource sharing T^+_{FPGA} and T^+_{lp} parameters are not proportional to the Table 3.5's ones, but still requiring aproximatelly > 5000 storage elements, a large quantity of resources but not prohibitive for hardware implementations.



(a)



(b)



(c)

Figure 3.12: Magnitude (a) and phase (b) frequency spectrum of optimum multi-stage decimation filter found by the optimization algorithm. (c) Passband ripple frequency spectrum.

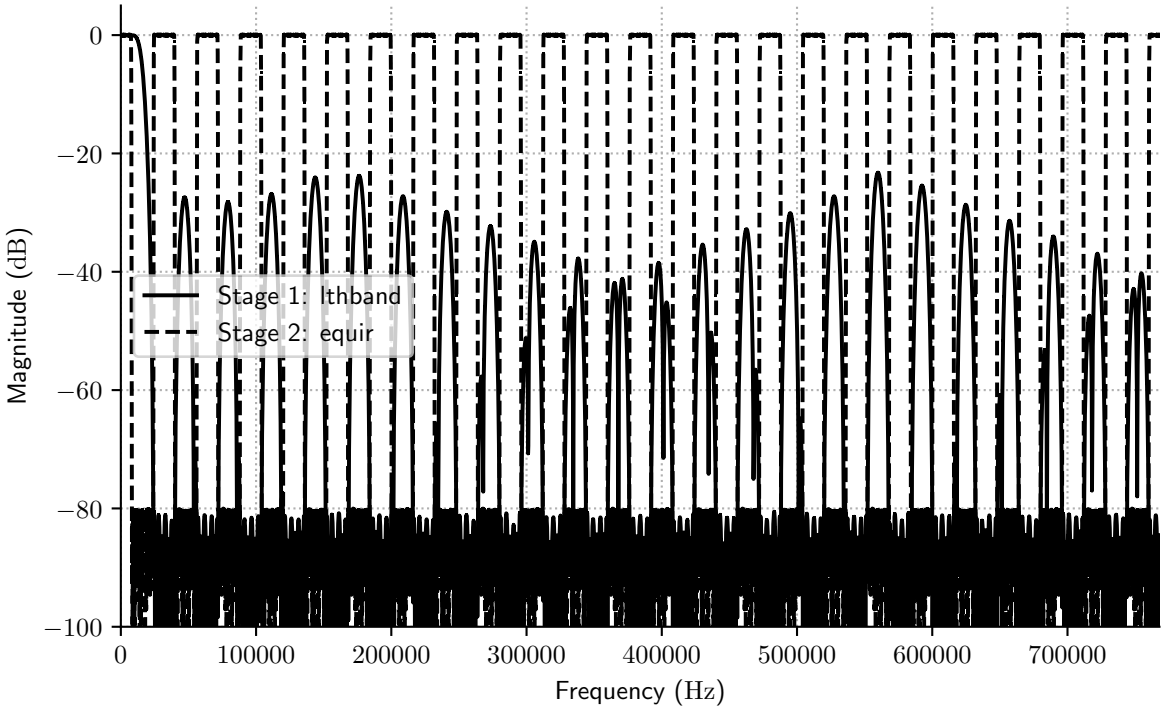


Figure 3.13: Magnitude frequency spectrum of the internal stages of the optimum multi-stage decimation filter found by the optimization algorithm.

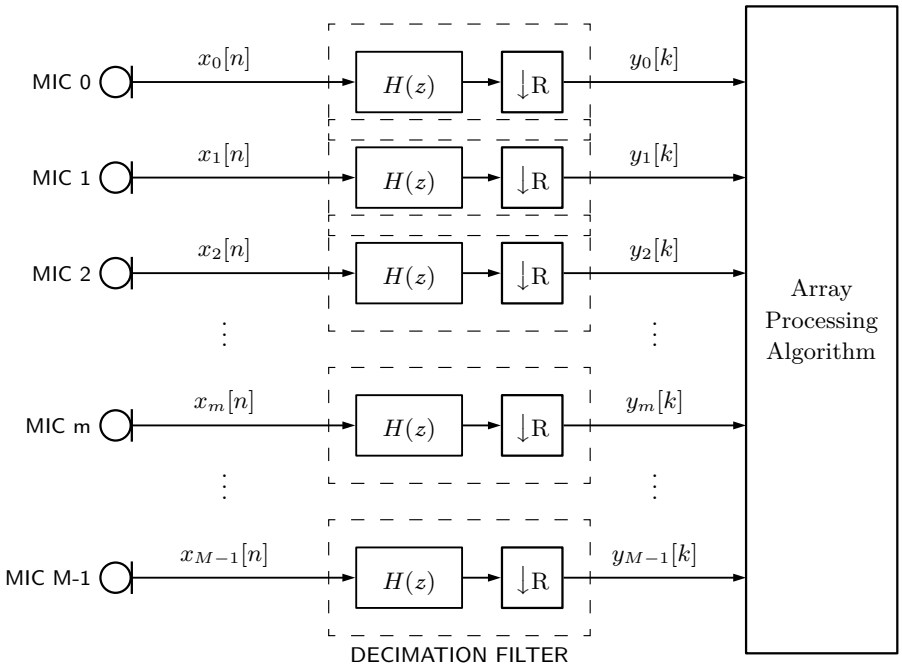


Figure 3.14: PDM-mic array processing system

	MS^{z}_{dec} (bit)	MS^{*}_{dec} (MPS)	MS^{+}_{dec} (APS)	MS^{o}_{dec} (APS)	f_{cpu} (MHz)	T^{+}_{FPGA} (-)	T^{+}_{lp} (-)
single_direct	1518000	1165885440000	2331648000000	3497533440000	3497533.44	54649	349754
single_eff	1518000	6072320000	12144000000	18216320000	18216.32	285	1822
single_memsav	15840	12144640000	12144000000	24288640000	24288.64	380	2429
multi_0	198520	1212800000	1210880000	4169600000	4169.60	66	417
multi_1	226760	769920000	765440000	4419840000	4419.84	70	442
multi_2	306640	159360000	784640000	4436480000	4436.48	70	444
multi_3	306640	159360000	784640000	4436480000	4436.48	70	444
multi_4	287080	139520000	1112960000	4543360000	4543.36	71	455
multi_5	230480	616320000	606720000	4693760000	4693.76	74	470
multi_6	254840	793600000	789120000	5048960000	5048.96	79	505
multi_7	231760	388480000	576000000	5052160000	5052.16	79	506
multi_8	241680	631040000	621440000	5078400000	5078.40	80	508
multi_9	229600	639360000	632320000	5132800000	5132.80	81	514

Table 3.6: Comparison of required resources to implement a PAPS using 40 multi-stage decimation filters found by the optimization algorithm in parallel.

Chapter 4

Beamforming at PCM and PDM domain

"Given a decimation filter and a beamformer specification, find an efficient beamforming implementation that works on the PDM domain."

In order to meet the aforementioned and previously formulated objective (Section 1.2.2), at first, this chapter reviews the mathematical basis and the state-of-the-art DAS beamformer implementation methods in time and frequency domains.

Then, time domain and frequency domain implementations that work on the PDM domain are proposed. It is shown that these implementations require only one decimation filter for all channels and they are more precise than the same implemented in PCM domain.

Finally, as these algorithms require decimation filters in their structure, the required implementation resources are estimated and compared to their analogous state-of-the-art implementations using optimized decimation filters structures calculated in Chapter 3 for Table 1.1 specification.

4.1 State-of-the-art: Beamforming at PCM domain

The DAS beamformer is the oldest and simplest array signal processing algorithm [4]. The underlying idea is to delay each microphone input by an appropriate time delay and then add all delayed microphone signals together. In this sense, the audio signal arriving from a certain direction at the array will be reinforced with respect to other signals arriving from other directions and incoherent noise.

The DAS algorithm can be implemented in time domain [43,44], using delay blocks and an adder; or in frequency domain using the FFT. In this section the mathematical basis of these time and frequency domain implementation methods will be presented.

4.1.1 Time domain implementations

Discrete-time beamformer

The traditional or discrete-time DAS beamformer[1] is the result of

$$z[k] = \sum_{m=0}^{M-1} w_m y_m[k - k_m] \,, \tag{4.1}$$

where y_m is the mth microphone's output in PCM representation so that, in case of PDM-mics, y_m would be the decimation filter's output. In this sense, to avoid confusion, the PDM bitstream incoming from the respective mth digital microphone will be represented as x_m. Also, in (4.1), k_m is the integer delay associated to the mth microphone so that

$$k_m = \|\Delta_m / T\| = \|\Delta_m f_o\| \,,$$

where Δ_m is the required delay in the mth microphone, $\|x\|$ means the nearest integer to x, and f_o and T are the sampling rate and period in y_m respectively. Equation 4.1 can be implemented as shown in the block diagram in Figure 1.1 using PDM-mics.

Also, due to the integer nature of k_m, the *discrete-time beamformer* does not allow to form sums that involve non integer multiples of T. Consequently, beams cannot be steered in arbitrary directions. Figure 4.6a shows the normalized power of a 40-microphone uniform linear array implemented with the *discrete-time beamformer* method. Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength. The stepped response shown in the normalized power diagram is due to the integer nature of the delay elements which limits the beamformer resolution. The equations to estimate the required resources for this kind of beamformer are discussed in the end of this section.

Discrete-time interpolation beamformer

In order to overcome the limitations of the discrete-time beamformer involving non-integer multiples of T, each sensor input signal $y_m[k]$ can be interpolated i.e. the sensor signal $y_m[k]$ can first be upsampled, delayed, and then passed through a low-pass interpolation filter. The upsampling operation is intended as an interspersing of $I - 1$ zero-valued samples between sensor samples. This can be represented as

$$u'_m[k'] = \uparrow_I \{y_m[k]\} \,,$$

[1] In literature, the traditional DAS does not have the weights w_m in its temporal representation. These weights only show up if you use a "weighted DAS", or in the frequency representation. However, in this work the "weighted DAS" is referred as the traditional DAS.

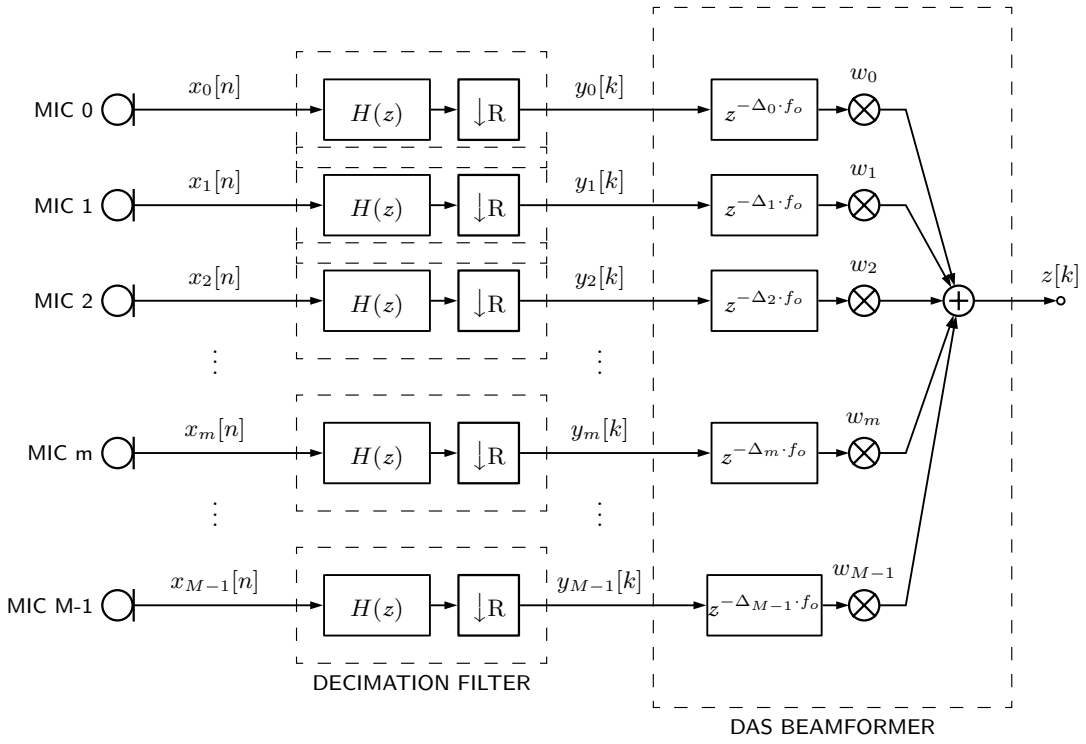


Figure 1.1: PDM-mic array DAS beamformer (repeated from page 23)

where $\uparrow_I \{\cdot\}$ represents an upsampling by I operation so that $u'_m[k']$ is a signal at sampling period $T' = T/I$. Thus, the interpolated signal after the low-pass interpolation filter with impulse response $h_I[k']$ will yield

$$y'_m[k'] = h_I[k'] * u'_m[k'] \,,$$

where $*$ represents discrete-time convolution operation. In practice, if $h_I[k']$ is properly designed, $y'_m[k']$ is just as if $y_m[k]$ would be sampled at a higher sampling rate.

All interpolated signals can be delayed, weighted and summed together as shown in Figure 4.2 to produce

$$z'[k'] = \sum_{m=0}^{M-1} w_m y'_m[k' - k'_m] \,, \tag{4.2}$$

where

$$k'_m = \|\Delta_m / T'\| = \|I \Delta_m f_o\|$$

is the required integer delay in the interpolated signals. See that, although this delay is still integer as k_m case, due to the new sampling period T' rather than T, it delivers finer resolution. Also, as the beamformer output still needs to be at sampling period T, $z'[k']$ needs to be downsampled by I at the end. Therefore,

$$z[k] = \downarrow_I \{z'[k']\} \,,$$

where $\downarrow_I \{\cdot\}$ represents a downsampling by I operation. Because the delay and sum are performed in the interpolated signals, we call this method as *discrete-time interpolation beamformer*. Figure 4.6b shows the normalized power of a 40-microphone uniform linear array implemented with the *discrete-time beamformer* method. Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength. This figure shows that the interpolated beamformer has a finer resolution than the *discrete-time beamformer* because of increased number of delay elements at a higher sampling rate. The equations to estimate the required resources for this kind of beamformer are discussed at the end of this section.

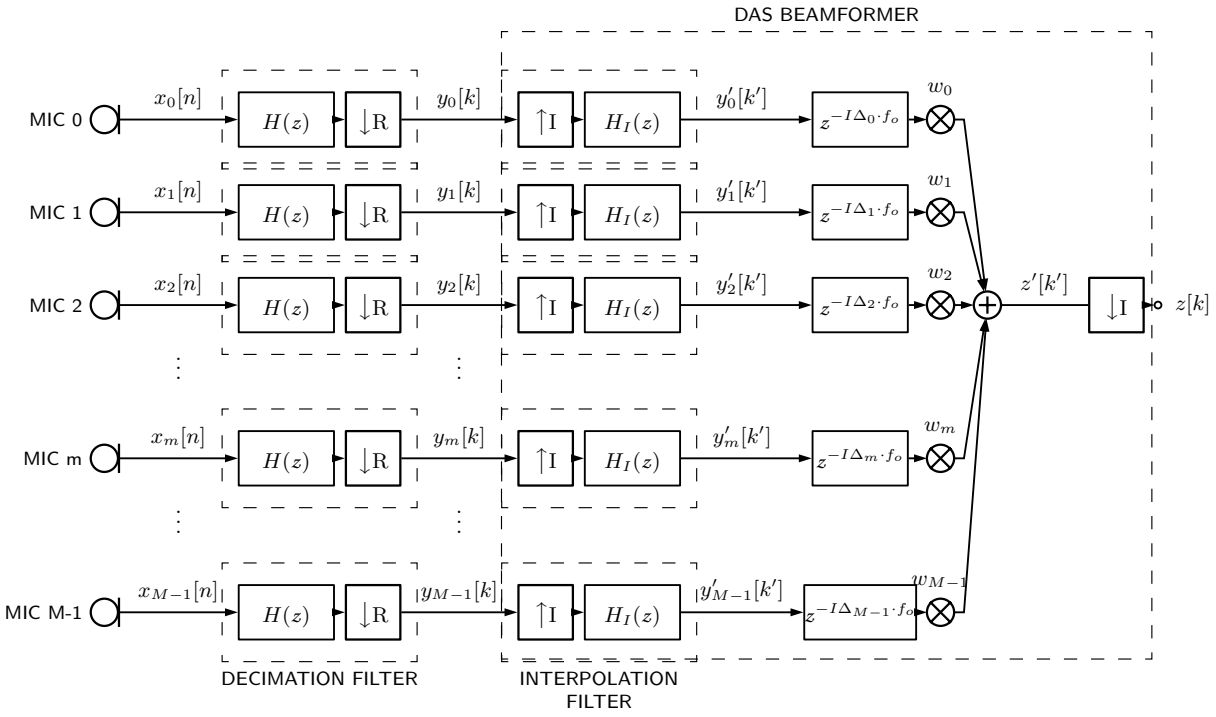


Figure 4.2: Discrete-time interpolation beamformer.

In order to reduce calculations by a factor of M, the interpolation filter can be expressed in its so-called *polyphase* filter structure [13, 14] in a similar way explained for a decimation filter in Section 2.3.2 as shown in Figure 4.3. In this sense, given the interpolation filter $h_I[k']$, as $k' = kI + r \; \forall r \in \{0, \ldots, I-1\}$, its output can be expressed as

$$y'_m[k'] = y'_m[kI + r] = \sum_s y_m[s] h_r[k - s] \,, \tag{4.3}$$

where

$$h_r[k] = h_I[kI + r] \,, \tag{4.4}$$

is the polyphase component.

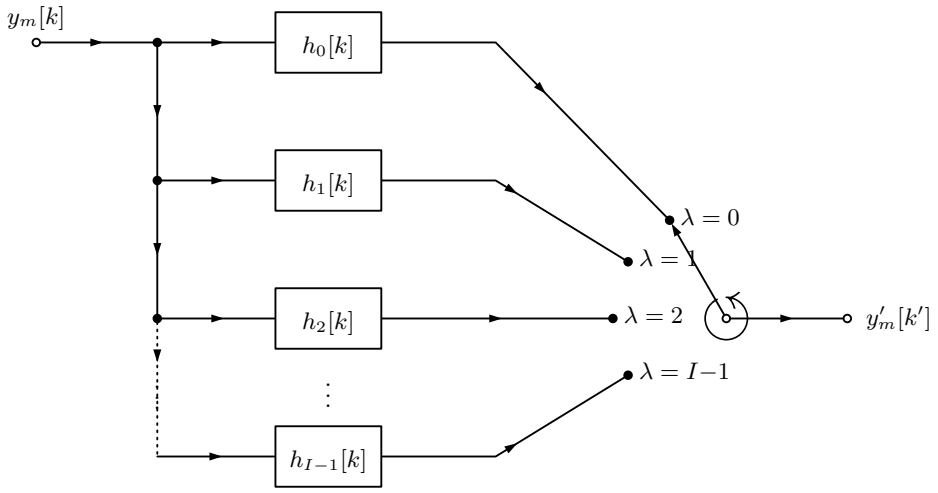


Figure 4.3: Interpolation filter in polyphase filter structure.

Replacing (4.3) and (4.4) in (4.2)

$$z'[kI + r] = \sum_{m=0}^{M-1} w_m \left[\sum_s y_m[s] h_I[(k-s)I + r - k'_m] \right]. \tag{4.5}$$

It is defined

$$h_{(r-k'_m)_I}[k] = h_I[kI + r - k'_m],$$

where computing the difference $r - k'_m$ modulo I means bringing the difference back into the range $[0, I-1]$ by adding or subtracting multiples of I from k'_m and delaying or advancing $h_{r-k'_m}$ by the appropriate integer number of samples. Equation (4.5) can be written as

$$z'[kI + r] = \sum_{m=0}^{M-1} w_m \left[\sum_s y_m[s] h_{(r-k'_m)_I}[k-s] \right]. \tag{4.6}$$

Because the beamformer output needs to be at the same sampling rate than the input, it is required that $z'[kI + r]$ would be downsampled by I. In this sense, it is assumed a fixed value of $r = 0$ so that (4.6) could be written finally as

$$z[k] = z'[kI] = \sum_{m=0}^{M-1} w_m \left[\sum_s y_m[s] h_{(-k'_m)_I}[k-s] \right]. \tag{4.7}$$

Equation (4.7) represents an efficient polyphase implementation of the discrete-time interpolation beamformer. As shown in Figure 4.4, in this implementation, for each beam, only one of the component polyphase filters needs to be implemented at each sensor's output.

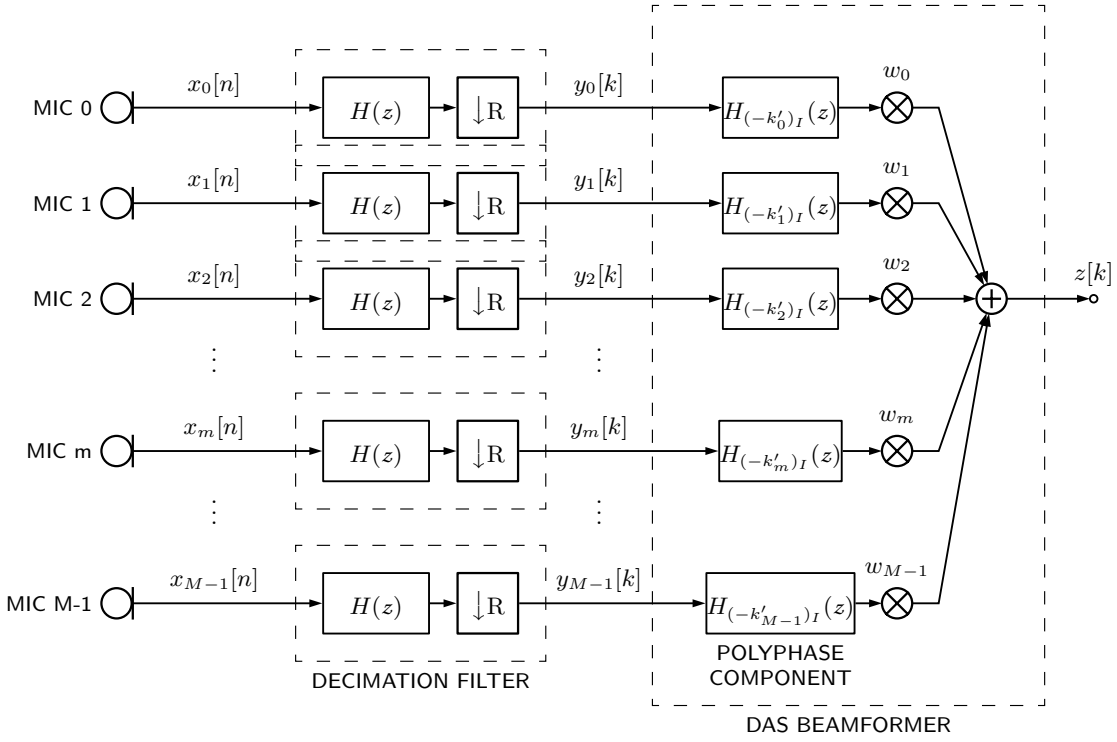


Figure 4.4: Efficient discrete-time interpolation beamformer.

Discrete-time *postdecimation* interpolation beamformer

Because of linearity, low-pass filter and beamforming operations can be interchanged as shown in Figure 4.5. This implementation results in computational savings when the number of sensors M is significantly larger than the interpolation factor I.

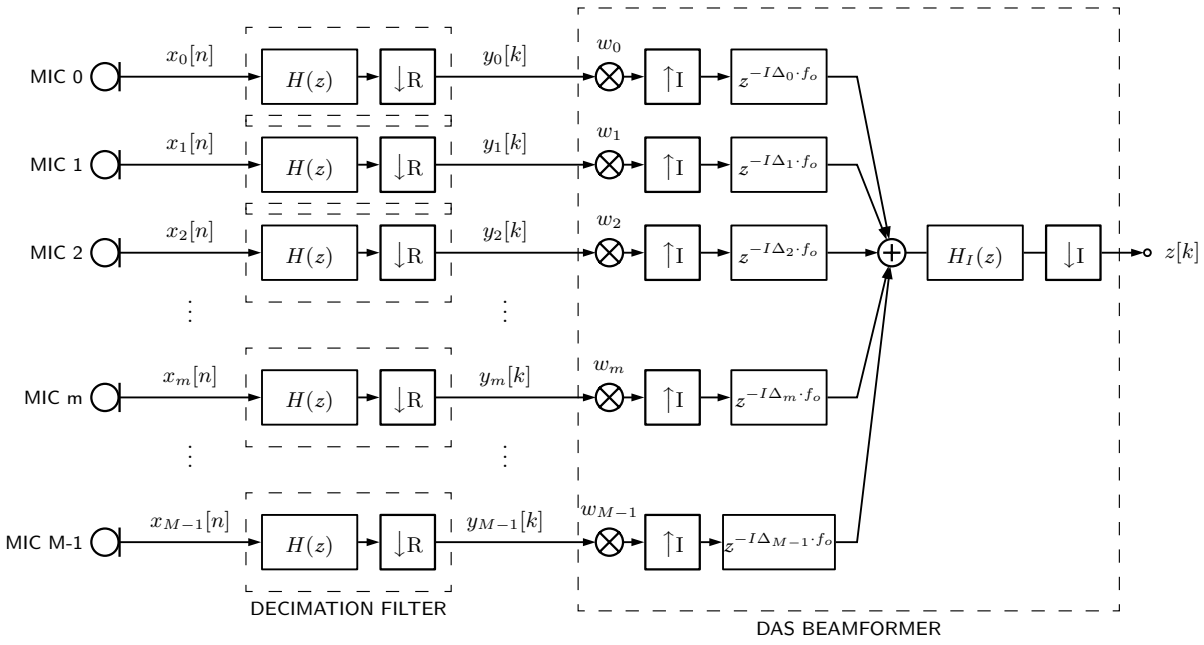
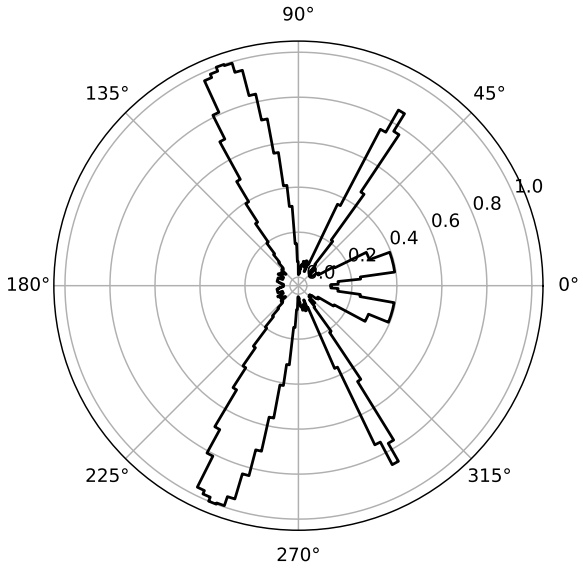
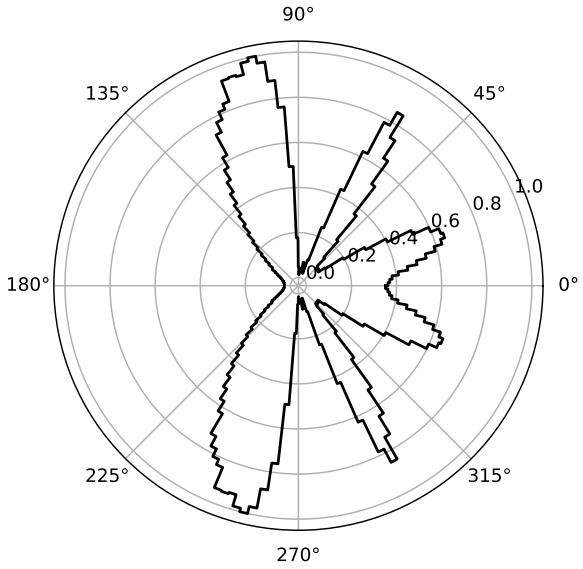


Figure 4.5: Discrete-time *postdecimation* interpolation beamformer.



(a) Discrete-time beamformer method.



(b) Discrete-time interpolation beamformer method (interpolation factor $I = 16$).

Figure 4.6: Normalized power (polar) of a uniform linear array of 40 microphones ($M = 40$) and specifications as listed in Table 1.1 and Table 1.2. Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength.

Implementation resources

The implementation resources for the time domain beamformers presented in this section can be derived from Figure 1.1, 4.4 and 4.5 as shown in Table 4.1. In this table, S^z_{dec}, S^+_{dec}, S^*_{dec} and S^o_{dec} are the implementation resources of each decimation filter.

Also, as delay from the array center to the mth microphone (Δ_m) will be changing to steer the beamformer, the delay chains need to be large enough support all possible values of Δ_m. Therefore, from (1.4) is derived that the delay chains length for the discrete-time beamformer will be $2\lceil\Delta_{\max} f_o\rceil$; and for the *efficient* and *postdecimation* ones, $2\lceil I\Delta_{\max} f_o\rceil$ as shown in Table 4.1. Note also that all quantities are added by M times S^z_{dec}, S^+_{dec} or S^*_{dec} respectively, as the decimation filter are dedicated by channel, not sharing any resource.

It is also observed that both interpolated realizations depend on the interpolation filter length (N_I), the *efficient* one has the $\lceil N_I/I\rceil$ term in its calculations because of the polyphase structure used.

Finally, comparing S^*_{dec} from *efficient* and *postdecimation* interpolation beamformers, it can be verified that the *postdecimation* one is more resource-efficient only when $M\lceil\frac{N_I}{I}\rceil \geq N_I I$ or approximately when $M > I^2$.

Implementation Method	**Beamformer's storage requirement ($S^z_{\mathbf{bf}}$) in** bit	**Beamformer's number of additions per second ($S^+_{\mathbf{bf}}$) in** APS	**Beamformer's number of multiplications per second ($S^*_{\mathbf{bf}}$) in** MPS
Discrete-time beamformer	$2M\lceil\Delta_{\max} f_o\rceil L_{\text{out}} + MS^z_{\text{dec}}$	$(M-1)f_o + MS^+_{\text{dec}}$	$Mf_o + MS^*_{\text{dec}}$
Efficient discrete-time interpolation beamformer	$2M(\lceil I\Delta_{\max} f_o\rceil + \lceil\frac{N_I}{I}\rceil)L_{\text{out}} + MS^z_{\text{dec}}$	$(M-1+M\lceil\frac{N_I}{I}\rceil)f_o + MS^+_{\text{dec}}$	$(M + M\lceil\frac{N_I}{I}\rceil)f_o + MS^*_{\text{dec}}$
Discrete-time *postdecimation* interpolation beamformer	$(2M\lceil I\Delta_{\max} f_o\rceil + N_I)L_{\text{out}} + MS^z_{\text{dec}}$	$(M-1+N_I)f_o I + MS^+_{\text{dec}}$	$(M+N_I I)f_o + MS^*_{\text{dec}}$

Table 4.1: Time domain implementation resources of beamformers at PCM domain

4.1.2 Frequency domain implementations

One-dimensional FFT beamformer

Given an array of M microphones with time domain outputs and denoting the mth microphone output as $y_m(t)$, we denote the Fourier transform of the mth microphone output by $Y_m(\omega)$. In this case the spectrum of the delay-and-sum beamformer output would be

$$Z(\omega) = \sum_{m=0}^{M-1} w_m Y_m(\omega)\exp(-j\omega\Delta_m)\,, \tag{4.8}$$

where Δ_m is the delay in the mth microphone output $y_m(t)$. In practice, however, $Y_m(\omega)$ can not be computed because it would require integrating over all time. So, in order to analyze the time domain signal in a limited time frame, it is introduced the concept of short-time Fourier transform

$$Y_m(t,\omega) = \int_t^{t+D} \psi(t-\tau)y_m(\tau)e^{-j\omega\tau}d\tau\,, \tag{4.9}$$

where D is the time frame after the t instant. Here, $\psi(t)$ denotes a finite length window defined over $[0, D]$. Therefore, (4.9) can be rewritten as

$$Y_m(t,\omega)e^{j\omega t} = \int_0^D \psi(\tau) y_m(t+\tau) e^{-j\omega\tau} d\tau \,. \tag{4.10}$$

This expression can be interpreted as an approximation of the spectrum at time t in a frame of length D. Then, the spectrum of the delay-and-sum beamformer output will be

$$Z(t,\omega) = \sum_{m=0}^{M-1} w_m Y_m(t,\omega) e^{j\omega t} \exp(-j\omega\Delta_m) \,. \tag{4.11}$$

Even though (4.11) is limited in time, it is still required to integrate over the time domain which is not possible for discrete-time signal processing. Provided that $y_m[k]$ is the kth sample of $y_m(t)$ signal sampled at $f_o = 1/T$ rate so that $t = kT$, (4.9) can be expressed in the discrete-time domain as

$$Y_m[k,\omega] = \sum_{l=k}^{k+D_s-1} \psi[l-k] y_m[l] e^{-j\omega T l} \,, \tag{4.12}$$

where $Y_m[k,\omega]$ is the discrete short-time Fourier transform of the $y_m(t)$ at the instant $t = kT$ and over the frame time $D = D_s T$ provided that D_s is an integer. Here, $\psi[k]$ is also the discrete-time version of the window function $\psi(t)$ over $[0, D_s]$. Then the discrete short-time Fourier transform of the beamformer output equals to

$$Z[k,\omega] = \sum_{m=0}^{M-1} w_m Y_m[k,\omega] e^{j\omega T k} \exp(-j\omega\Delta_m) \,. \tag{4.13}$$

If the frequency domain is discretized so that $\omega T = 2\pi v/D_s$ for $v = 0, \ldots, D_s - 1$, (4.12) can be rewritten as

$$Y_m[k,v] \exp\left\{ j\frac{2\pi v}{D_s} k \right\} = \sum_{l=0}^{D_s-1} \psi[l] y_m[k+l] \exp\left\{ -j\frac{2\pi v}{D_s} l \right\} . \tag{4.14}$$

So the beamformer output will be

$$Z[k,v] = \sum_{m=0}^{M-1} w_m Y_m[k,v] \exp\left\{ j\frac{2\pi v}{D_s}(k - \Delta_m/T) \right\} . \tag{4.15}$$

Then, if it is defined $\tilde{y}_m[k,l] = \psi[l] y_m[k+l]$, the DFT of $\tilde{y}_m[k,l]$ will be

$$\tilde{Y}_m[k,v] = \sum_{l=0}^{D_s-1} \tilde{y}_m[k,l] \exp\left\{ -j\frac{2\pi v}{D_s} l \right\} , \qquad v = 0, \ldots, D_s - 1 \,. \tag{4.16}$$

Finally, Equations (4.14) and (4.15) can be rewritten as

$$Y_m[k,v]\exp\left\{j\frac{2\pi v}{D_s}k\right\} = \tilde{Y}_m[k,v]\,, \qquad v = 0,\ldots,D_s-1\,, \tag{4.17}$$

$$Z[k,v] = \sum_{m=0}^{M-1} w_m \tilde{Y}_m[k,v]\exp\left\{-j\frac{2\pi v}{D_s}\frac{\Delta_m}{T}\right\}\,, \qquad v = 0,\ldots,D_s-1\,. \tag{4.18}$$

Equation (4.18) can be implemented in hardware as shown in Figure 4.7. At first, it is required to transform PDM bitstreams $x_m[n]$ to PCM representation $y_m[k]$. Then each PCM audio signal is passed through a windowing function $\psi[k]$ to yield $\tilde{y}_m[k]$. Each windowed signal is then passed through a FFT block to obtain $\tilde{Y}_m[k,v]$. These frequency domain signals should be multiplied by a weighting factor

$$W_m(v) = w_m \exp\left\{-j\frac{2\pi v}{D_s}\frac{\Delta_m}{T}\right\}$$

which depends on the desired direction of arrival. Finally, the weighted outputs are summed together and transformed to time domain by an Inverse Fast Fourier transform (IFFT).

Figure 4.9a shows the normalized power of a uniform linear array implemented with *one-dimensional FFT beamformer* method using 40 microphones ($M = 40$). Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength. The equations to estimate the required resources for this kind of beamformer are discussed in the end of this section.

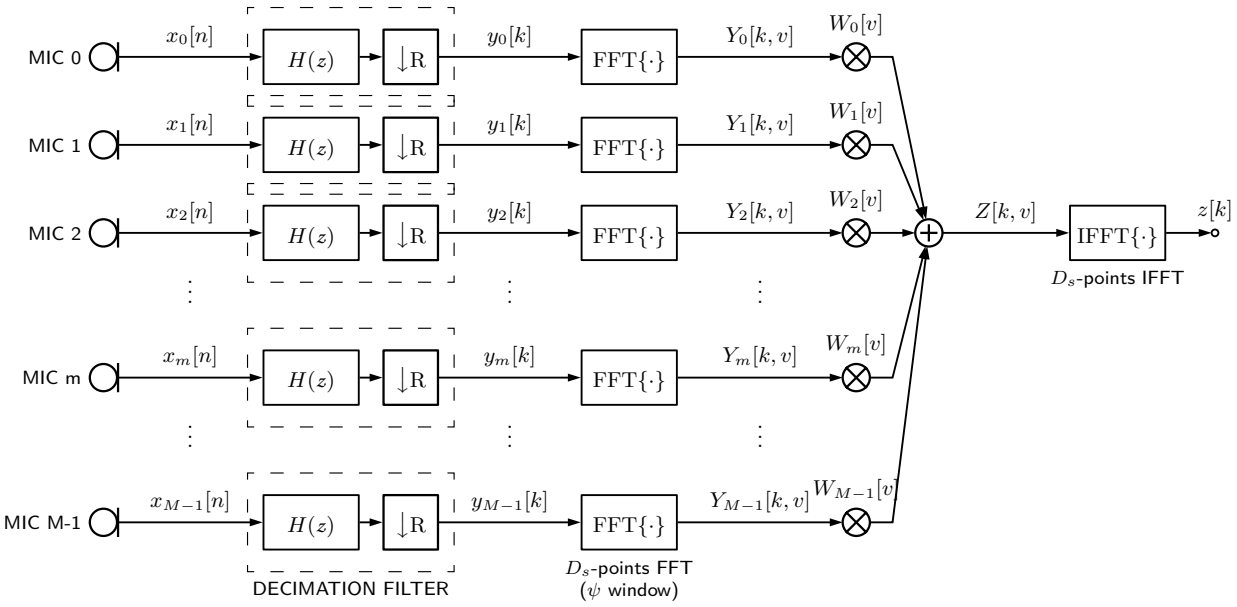


Figure 4.7: One-dimensional FFT beamformer implementation method.

Two-dimensional FFT beamformer

In the same way than the *one-dimensional FFT beamformer*, if we use a uniform regular array whose spatial origin occurs at the first microphone such that

$$\Delta_m = \frac{\alpha_x m d}{c}\,;$$

where d is the space between microphones, $\alpha_x = \cos(\theta)$, θ is the angle of arrival and c is the sound speed.

If it is also defined a variable u related to v as

$$u = \frac{dM\alpha_x}{D_s T c} v\,, \quad v = 0, \ldots, D_s - 1\,, \tag{4.19}$$

such that the argument of the exponential of (4.18) can be rewritten as

$$\frac{2\pi v}{D_s}\frac{\Delta_m}{T} = \frac{2\pi u}{M} m\,, \qquad m = 0, \ldots, M-1\,,$$

and the whole equation (4.18), replacing (4.19) and (4.16), can be rewritten as following

$$Z[k,u,v] = \sum_{m=0}^{M-1}\sum_{l=0}^{D_s-1} w_m \tilde{y}_m[k,l] \exp\left\{-j\frac{2\pi v}{D_s} l\right\} \exp\left\{-j\frac{2\pi u}{M} m\right\}\,. \tag{4.20}$$

And if it is also defined $x[k,m,l] = w_m\tilde{y}_m[k,l] = w_m\psi(l)y_m[k+l]$, thus (4.20) can be written as a two-dimensional DFT

$$Z[k,u,v] = \mathrm{DFT}\left\{\mathrm{DFT}\left\{x[k,m,l]\right\}\right\}\,, \qquad m = 0, \ldots, M-1\; l = 0, \ldots, D_s - 1\,. \tag{4.21}$$

Equation (4.21) can be implemented in hardware as shown in Figure 4.8. At first, it is required to transform PDM bitstreams $x_m[n]$ to PCM representation $y_m[k]$. Then each PCM audio signal is passed through a windowing function $\psi[k]$ to yield $\tilde{y}_m[k]$. Each windowed signal is then passed through a FFT block to obtain $\tilde{Y}_m[k,v]$. The outputs of those FFT blocks are passed through another FFT block. The output of this second FFT block is passed through a steerer block $S\{\cdot\}$ defined as

$$Z[k,v] = S(\theta, Z[k,u,v]) = Z[k,u,v]\Big|_{u=\frac{dM\cos(\theta)}{D_s T c}v}\,; \tag{4.22}$$

which, given a desired angle of arrival θ, filters only the samples meeting the u and v relation defined by (4.19). Finally, the Steerer's output is passed through an IFFT block.

Figure 4.9b shows the normalized power of a 40-microphone uniform linear array implemented with the *two-dimensional FFT beamformer* method. Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three

ones with equal strength. The stepped response shown in the normalized power diagram is due to the small size of the second FFT block ($N = 40$) which limits the beamformer resolution. This resolution can be improved increasing N so that it would be always greater or equal than the number of microphones ($N \geq M$).The equations to estimate the required resources for this kind of beamformer are discussed in the end of this section.

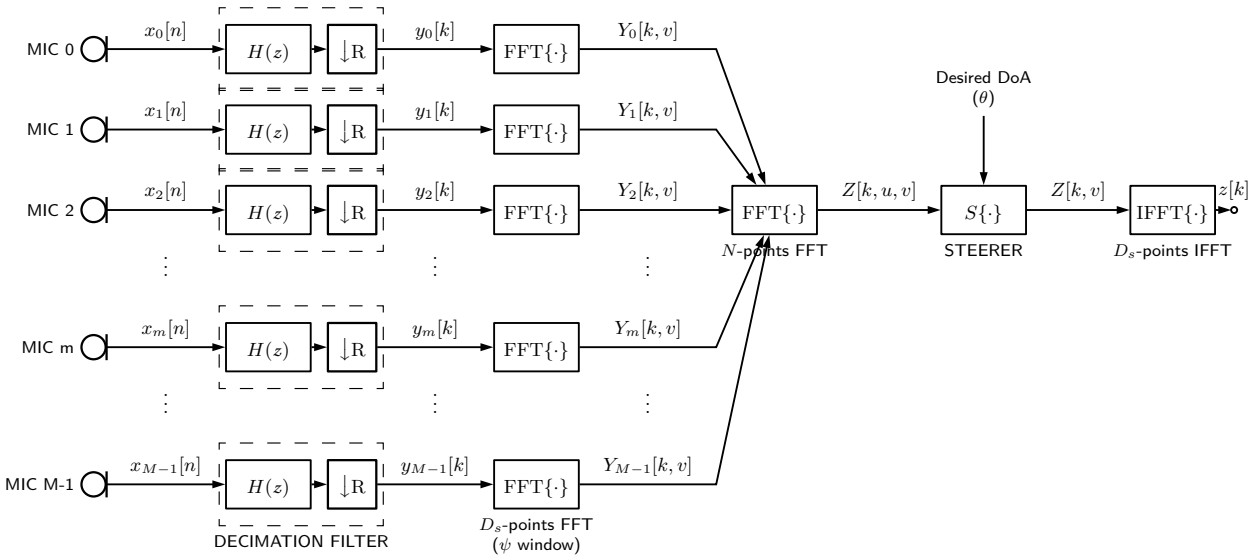


Figure 4.8: Two-dimensional FFT beamformer implementation method.

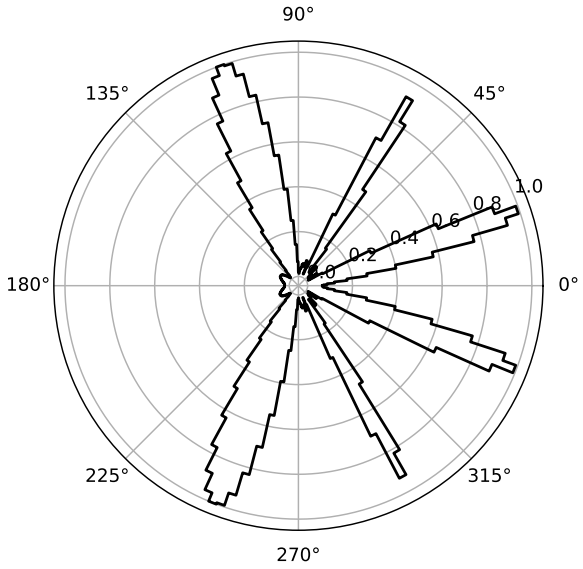
Implementation resources

If the D_s-points FFT blocks in Figures 4.7 and 4.8 are implemented using "butterfly" structures [13], each FFT block implementations resources can be estimated using (2.18). So, using this equation, it is found that each D_s-points FFT block will require $3D_sL_{\text{out}}$ storage elements, D_sL_{out} to store the input elements and $2D_sL_{\text{out}}$ to store each stage results (imaginary and real parts); $2D_s \log_2(D_s) f_o$ additions and multiplications per second. The same implementation metrics can be found for N-points FFT and D_s-points IFFT blocks with the same equation (2.18), assuming that they are also implemented with "butterfly" structures,

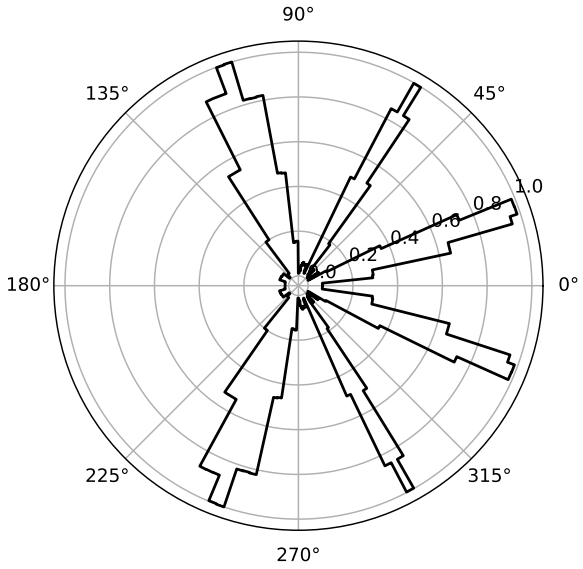
Based on these architectural assumptions, the implementation resources of one-dimensional and two-dimensional FFT beamformers can be calculated as shown in Table 4.2 where S^z_{dec}, S^+_{dec} and S^*_{dec} are the implementation resources of each decimation filter in the beamformer; and $N \geq M$ is the number of points of the 2^{nd} FFT in the two-dimensional FFT beamformer.

From Table 4.2 can be observed by comparison that the two-dimensional FFT beamformer requires less computation (MPS) when

$$D_sM \geq N \log_2 N \,,$$



(a) One-dimensional FFT method



(b) Two-dimensional FFT method

Figure 4.9: Normalized power (polar) of a uniform linear array of 40 microphones ($M = 40$) and specifications as listed in Table 1.1 and Table 1.2. Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength.

but as $N \geq M$, then two-dimensional FFT requires less computation when

$$N \leq 2^{D_s} .$$

Finally, it is observed also that even with less computation, two-dimensional FFT beamformer require additional $4NL_{\text{out}}$ storage elements. So, because of this storage and

Implementation Method	**Beamformer's storage requirement ($S^z_{\mathbf{bf}}$) in** bit	**Beamformer's number of additions per second ($S^+_{\mathbf{bf}}$) in** APS	**Beamformer's number of multiplications per second ($S^*_{\mathbf{bf}}$) in** MPS
One-dimensional FFT beamformer	$3MD_sL_{\text{out}} + 4D_sL_{\text{out}} + MS^z_{\text{dec}}$	$2(M+1)D_s \log_2 D_s f_o + 2D_s(M-1)f_o + MS^+_{\text{dec}}$	$2(M+1)D_s \log_2 D_s f_o + 2D_sMf_o + MS^*_{\text{dec}}$
Two-dimensional FFT beamformer	$3MD_sL_{\text{out}} + 4NL_{\text{out}} + 4D_sL_{\text{out}} + MS^z_{\text{dec}}$	$2(M+1)D_s \log_2 D_s f_o + 2N \log_2 N f_o + MS^+_{\text{dec}}$	

Table 4.2: Frequency domain implementation resources of beamformers at PCM domain

computation resources trade-off, the implementation should be chosen depending on the intended application and the available resources.

4.2 Proposal: Beamforming at PDM domain

In this section, an alternative time domain implementation and two frequency domain methods that do not require decimation filters are proposed. This chapter complements the study already done in [8] about beamforming on the PDM domain.

4.2.1 Time domain implementations

Discrete-time bitstream beamformer

The PCM signal y_m on the mth decimation filter's output can be written as

$$y_m[k] = \downarrow_R \{h[n] * x_m[n]\} = \sum_r h[r] x_m[Rk - r]\,, \tag{4.23}$$

where $\downarrow_R \{\cdot\}$ represents a downsampling by R operation ($n = Rk$), $*$ represents a discrete-time convolution, $h[n]$ is the impulse response of the low-pass filter so that its cutoff frequency would be $\omega_c = \pi/R$, and $x_m[n]$ is the PDM bitstream signal incoming from the mth digital microphone.

The discrete-time DAS beamformer can be expressed in terms of PDM bitstream signal x_m replacing (4.23) in (4.1)

$$z[k] = \sum_{m=0}^{M-1} w_m \sum_r h[r] x_m[Rk - Rk_m - r]\,.$$

Since $h[r]$ does not depend on m, this equation can be written as

$$z[k] = \sum_r h[r] \sum_{m=0}^{M-1} w_m x_m[Rk - Rk_m - r]\,. \tag{4.24}$$

By defining $z'[n]$ as

$$z'[n] = \sum_{m=0}^{M-1} w_m x_m[n - n'_m]\,, \tag{4.25}$$

where $n'_m = Rk_m$ is the delay in the mth PDM bitstream x_m, the beamformer output $z[k]$ can be expressed in terms of $z'[n]$ replacing (4.25) in (4.24) so that

$$z[k] = \sum_r h[r] z'[Rk - r] = \downarrow_R \{h[n] * z'[n]\}\,. \tag{4.26}$$

Equation (4.26) can be implemented in hardware as shown in Figure 1.2. Therefore, as the beamforming is performed in the first filtering stage, directly into the bitstreams,

this beamformer implementation will be called as *discrete-time bitstream beamformer*. Also note that as the delay and weighting are conducted at PDM domain, there will be a finer resolution for the delay.

Finally, as the delayed inputs $x_m[n - n'_m]$ are still bitstreams, the multiplying by w_m operation will be actually rather a substitution operation. This substitution operation will increase the number of bits of the adder inputs. Also, as the adder output will be a multi-bit signal, LPF should be designed to have also a multi-bit input rather than a bitstream input like the *discrete-time beamformer* case. This variation in the LPF's input will not impact in the final output as it is always a multi-bit (PCM) output. Figure 4.11 shows the normalized power of a 40-microphone uniform linear array implemented with the *discrete-time beamformer* method. Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength. It is shown that this beamformer has a better resolution than the normalized power response of the beamformers shown in Figure 4.6 because of the higher resolution in the delay elements running at input sampling rate. The equations to estimate the required resources for this kind of beamformer are discussed in the end of this section.

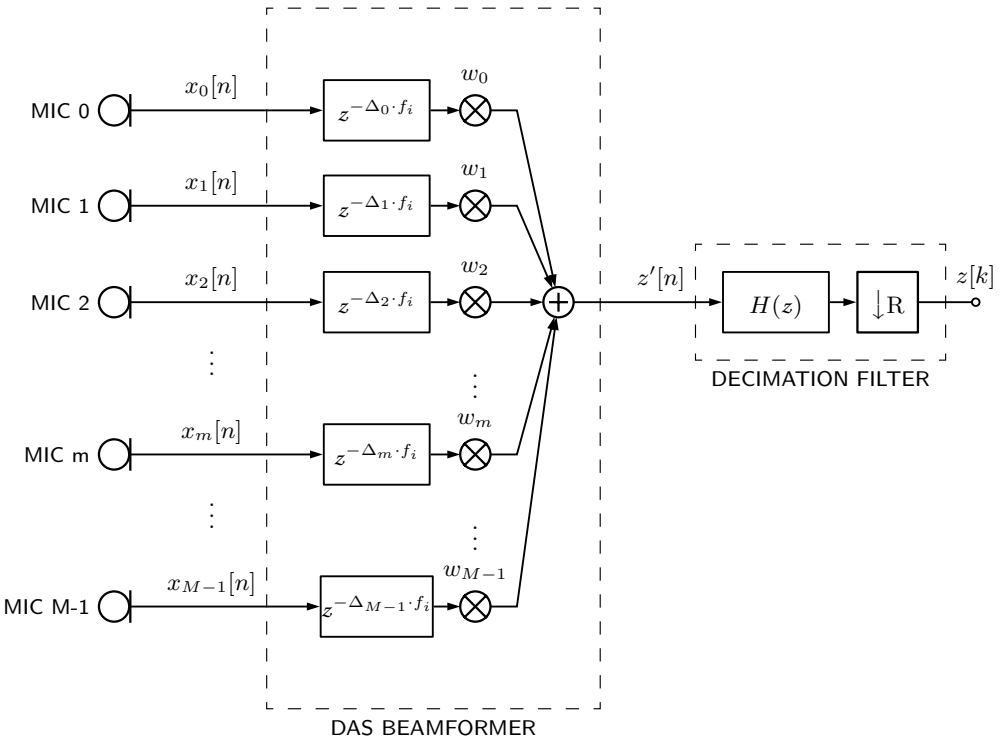


Figure 1.2: PDM-mic array DAS beamformer at PDM domain (repeated from page 25)

Implementation resources

The implementation resources for the *postdecimation* beamformer in Table 4.3 are derived from Figure 1.2. As the delay elements are in a higher sampling rate ($f_i = Rf_o$), the beamformer's storage requirement (S^z_{bf}) depends on R times $\Delta_{\max}$ and L_{in}. Also, as

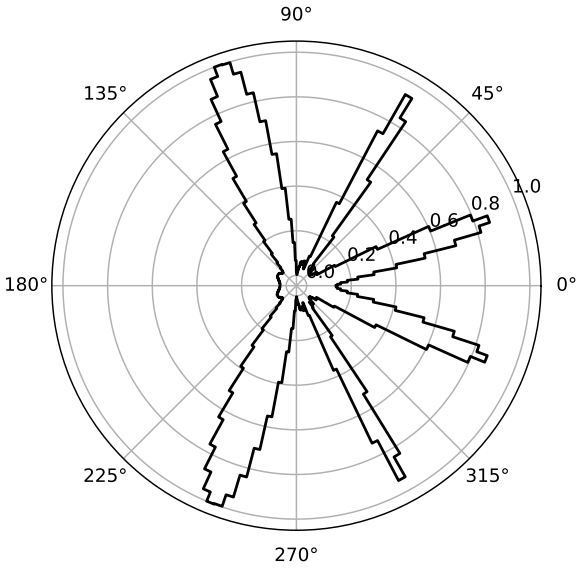


Figure 4.11: Discrete-time bitstream beamformer method. Normalized power (polar) of a uniform linear array of 40 microphones ($M = 40$) and specifications as listed in Table 1.1 and Table 1.2. Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength.

there is only a single decimation filter, implementation resources are added by S^z_{dec}, S^+_{dec} and S^*_{dec} respectively.

Implementation Method	**Beamformer's storage requirement ($S^z_{\mathbf{bf}}$) in** bit	**Beamformer's number of additions per second ($S^+_{\mathbf{bf}}$) in** APS	**Beamformer's number of multiplications per second ($S^*_{\mathbf{bf}}$) in** MPS
Discrete-time bitstream beamformer	$2M\lceil R\Delta_{\max} f_o \rceil L_{\text{in}} + S^z_{\text{dec}}$	$(M-1)Rf_o + S^+_{\text{dec}}$	$MRf_o + S^*_{\text{dec}}$

Table 4.3: Time domain implementation resources of beamformer at PDM domain

4.2.2 Frequency domain implementations

One-dimensional bitstream FFT beamformer

Due to the fact that a PDM bitstream has the same information than its converted PCM signal but with quantization noise shaped at higher frequencies, a PDM bitstream can be treated as a baseband signal with a higher sampling rate $f_i = Rf_o$, where f_o is the required sampling rate in the beamformer output and R is the decimation rate. Therefore, the *one-dimensional FFT beamformer* can be modified so that decimation would be performed in frequency domain at the end of the beamformer, before an IFFT is applied, as shown in Figure 4.12. This implementation method will be called as *one-dimensional bitstream FFT beamformer*.

Because the input sampling rate is higher than in a conventional DAS beamformer, it is required a frame length $D'_s = RD_s$ i.e. R times larger than the required in conventional implementation methods. Figure 4.14a shows the normalized power of a uniform linear array implemented with an *one-dimensional bitstream FFT beamformer* method using 40 microphones ($M = 40$). Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength. The equations to estimate the required resources for this kind of beamformer are discussed in the end of this section.

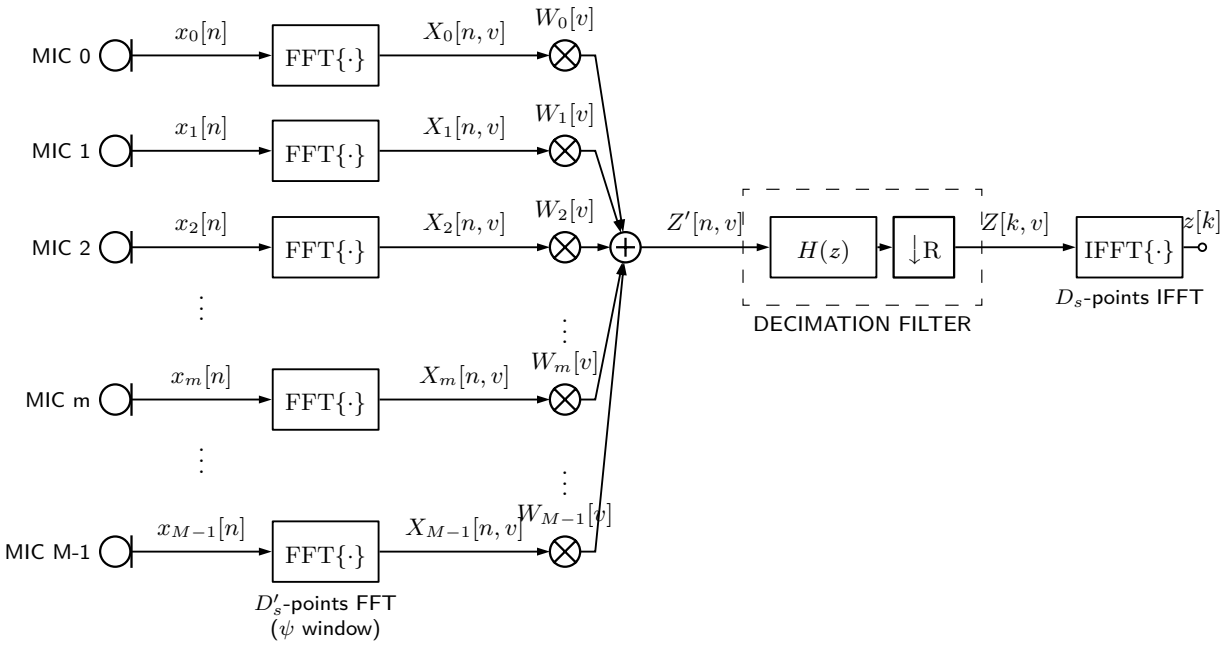


Figure 4.12: One-dimensional bitstream FFT beamformer implementation method.

Two-dimensional bitstream FFT beamformer

The *two-dimensional bitstream FFT beamformer* may be derived in the same way than the one-dimensional bitstream FFT beamformer, just performing the decimation filtering in the frequency domain at the end of the beamformer as shown in Figure 4.13. As the state-of-the-art beamformer case shown in Figure 4.8, the number of points of the second FFT block should have also $N \geq M$ points. See also that as IFFT operation takes place after the decimator, its number of points ($D'_s = D_s/R$) is less than the number of points in the first FFT block (D_s).

Figure 4.14b shows the normalized power of a uniform linear array implemented with a *two-dimensional bitstream FFT beamformer* method using 40 microphones ($M = 40$). Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength. The equations to estimate the required resources for this kind of beamformer are discussed in the end of this section.

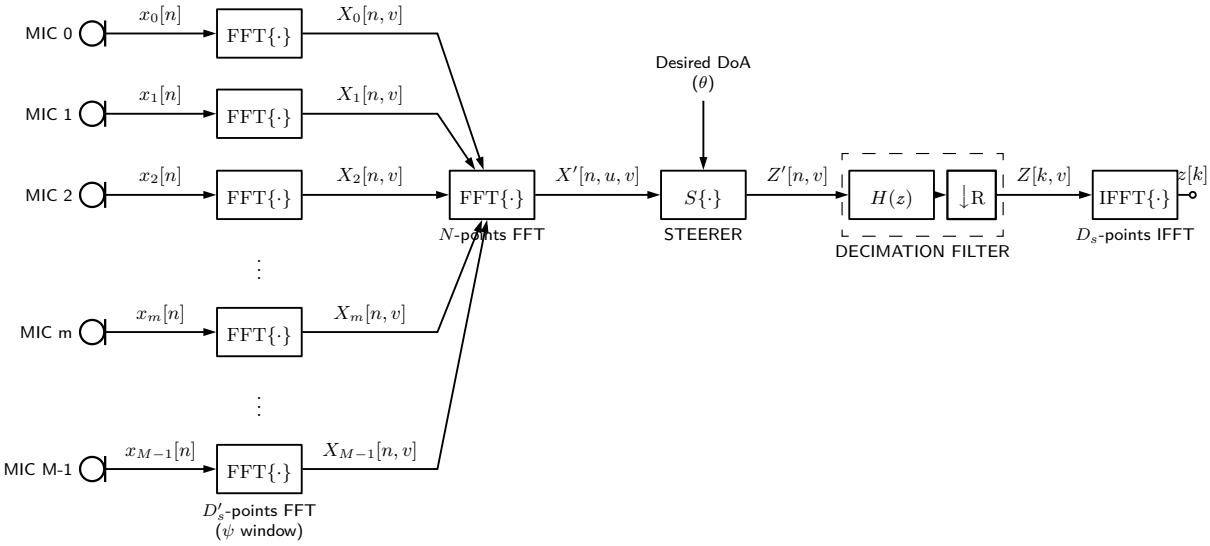


Figure 4.13: Two-dimensional bitstream FFT beamformer implementation method.

Implementation resources

If it is assumed that FFT and IFFT blocks from Figures 4.12 and 4.13 are implemented using "butterfly" structures, the required implementation resources for one-dimensional and two-dimensional bitstream FFT beamformers can be estimated as shown in Table 4.4 where the implementation resources of each FFT and IFFT are estimated using (2.18).

Implementation Method	**Beamformer's storage requirement ($S^z_{\mathbf{bf}}$) in** bit	**Beamformer's number of additions per second ($S^+_{\mathbf{bf}}$) in** APS	**Beamformer's number of multiplications per second ($S^*_{\mathbf{bf}}$) in** MPS
One-dimensional bitstream FFT beamformer	$3MRD_sL_{\text{out}} + 4D_sL_{\text{out}}$	$2M(RD_s)\log_2(RD_s)Rf_o + 2RD_s(M-1)Rf_o + 2D_s\log_2 D_sf_o$	$2M(RD_s)\log_2(RD_s)Rf_o + 2RD_sMRf_o + 2D_s\log_2 D_sf_o$
Two-dimensional bitstream FFT beamformer	$3MRD_sL_{\text{out}} + 4NL_{\text{out}} + 4D_sL_{\text{out}}$	$2M(RD_s)\log_2(RD_s)Rf_o + 2N\log_2 NRf_o + 2D_s\log_2 D_sf_o$	

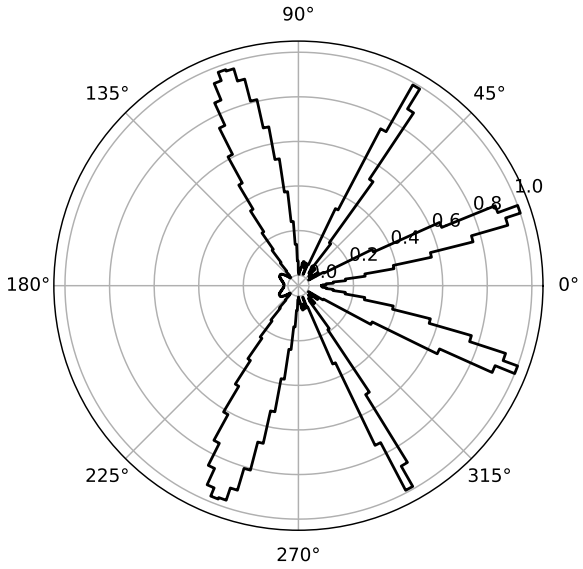
Table 4.4: Frequency domain implementation resources of beamformers at PDM domain

In the same way done for PCM beamformers in Section 4.2.2, by comparing the beamformer's number of multiplications per second (S^*_{bf}) and beamformer's storage requirement (S^z_{bf}) in Table 4.4, it can be observed that the two-dimensional bitstream FFT beamformer requires less multiplications per second when

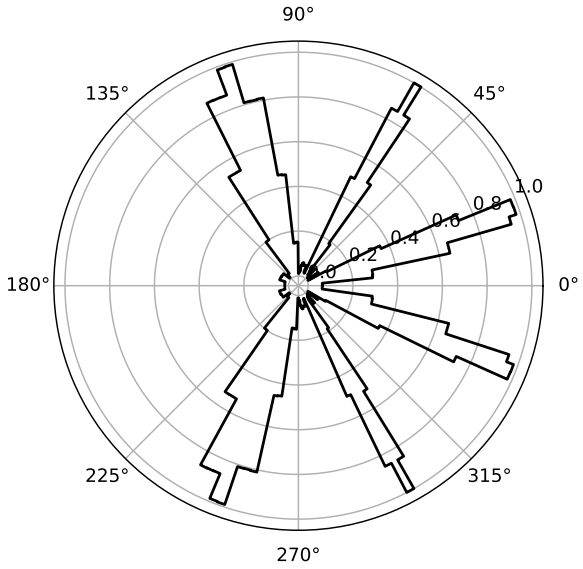
$$N \leq 2^{RD_s}$$

but requires additional $4NL_{\text{out}}$ bit than the one-dimensional one.

Finally, from Table 4.4 can be also observed that the decimation filters do not have impact in the resource utilization of both beamformer implementation, this because



(a) One-dimensional bitstream FFT method



(b) Two-dimensional bitstream FFT method

Figure 4.14: Normalized power (polar) of a uniform linear array of 40 microphones ($M = 40$) and specifications as listed in Table 1.1 and Table 1.2. Three audio sources of 1 kHz, 3 kHz and 5 kHz are located at 20, 60 and 110 degrees respectively, the three ones with equal strength.

decimation in frequency domain requires only to discard the higher frequency samples without any extra calculation.

4.3 Summary

Domain	Implementation Method	Beamformer's storage requirement ($S^z_{\mathbf{bf}}$) in bit	Beamformer's number of additions per second ($S^+_{\mathbf{bf}}$) in APS	Beamformer's number of multiplications per second ($S^*_{\mathbf{bf}}$) in MPS
Time	Discrete-time beamformer	$2M\lceil\Delta_{\max}f_o\rceil L_{\text{out}} + MS^z_{\text{dec}}$	$(M-1)f_o + MS^+_{\text{dec}}$	$Mf_o + MS^*_{\text{dec}}$
	Efficient discrete-time interpolation beamformer	$2M(\lceil I\Delta_{\max}f_o\rceil + \lceil\frac{N_I}{I}\rceil)L_{\text{out}} + MS^z_{\text{dec}}$	$(M-1+M\lceil\frac{N_I}{I}\rceil)f_o+MS^+_{\text{dec}}$	$(M + M\lceil\frac{N_I}{I}\rceil)f_o + MS^*_{\text{dec}}$
	Discrete-time *postdecimation* interpolation beamformer	$(2M\lceil I\Delta_{\max}f_o\rceil + N_I)L_{\text{out}} + MS^z_{\text{dec}}$	$(M-1+N_I)f_oI + MS^+_{\text{dec}}$	$(M+N_I I)f_o + MS^*_{\text{dec}}$
	Discrete-time bitstream beamformer[2]	$2M\lceil R\Delta_{\max}f_o\rceil L_{\text{in}} + S^z_{\text{dec}}$	$(M-1)Rf_o + S^+_{\text{dec}}$	$MRf_o + S^*_{\text{dec}}$
Frequency	One-dimensional FFT beamformer	$3MD_sL_{\text{out}} + 4D_sL_{\text{out}} + MS^z_{\text{dec}}$	$2(M+1)D_s\log_2 D_sf_o + 2D_s(M-1)f_o + MS^+_{\text{dec}}$	$2(M+1)D_s\log_2 D_sf_o + 2D_sMf_o + MS^*_{\text{dec}}$
	Two-dimensional FFT beamformer	$3MD_sL_{\text{out}} + 4NL_{\text{out}} + 4D_sL_{\text{out}} + MS^z_{\text{dec}}$	$2(M+1)D_s\log_2 D_sf_o + 2N\log_2 Nf_o + MS^+_{\text{dec}}$	
	One-dimensional bitstream FFT beamformer[3]	$3MRD_sL_{\text{out}} + 4D_sL_{\text{out}}$	$2M(RD_s)\log_2(RD_s)Rf_o + 2RD_s(M-1)Rf_o + 2D_s\log_2 D_sf_o$	$2M(RD_s)\log_2(RD_s)Rf_o + 2RD_sMRf_o + 2D_s\log_2 D_sf_o$
	Two-dimensional bitstream FFT beamformer[4]	$3MRD_sL_{\text{out}} + 4NL_{\text{out}} + 4D_sL_{\text{out}}$	$2M(RD_s)\log_2(RD_s)Rf_o + 2N\log_2 NRf_o + 2D_s\log_2 D_sf_o$	

Table 4.5: Beamformer implementation resources summary

In Table 4.5 is summarized the required resources to implement the state-of-the-art and proposal methods studied in this chapter. It is easy to see in this table that the *discrete-time beamformer* will require less implementation resources that the *interpolation* and *postdecimation* ones. So if it is assumed that $M \gg 1$ and the required resources for the *discrete-time beamformer* are compared to the proposed *discrete-time bitstream beamformer* one, the latter one will be more efficient that the former one when

$$S^+_{\text{dec}} > f_i - f_o, \tag{4.27a}$$

$$S^*_{\text{dec}} > f_i - f_o \quad \text{and} \tag{4.27b}$$

$$S^z_{\text{dec}} > 2\Delta_{\max}f_o(RL_{\text{in}} - L_{\text{out}})\,. \tag{4.27c}$$

In other words, the *discrete-time bitstream beamformer* will the most resource-efficient time domain beamformer when conditions in (4.27) are accomplished.

In frequency domain methods case, it is not possible to derive simple conditions to select the most resource-efficient time domain beamformer as they depend on the additional parameter D_s that depends on the frame length (L_{frame}) as follows:

$$D_s = \lceil L_{\text{frame}}f_o\rceil\,. \tag{4.28}$$

Also, depending on the FFT implementation method, it could be required D_s to be a multiple-by-2 number.

4.4 Results

Domain	Implementation Method	Beamformer's storage requirement ($S^{z}_{\mathbf{bf}}$) in bit	Beamformer's number of additions per second ($S^{+}_{\mathbf{bf}}$) in APS	Beamformer's number of multiplications per second ($S^{*}_{\mathbf{bf}}$) in MPS
Time	Discrete-time beamformer	210040	1211504000	1213440000
	Efficient discrete-time interpolation beamformer	302200	1213424000	1215360000
	Discrete-time *postdecimation* interpolation beamformer	297160	1221920000	1218240000
	Discrete-time bitstream beamformer	82323	150080000	153200000
Frequency	One-dimensional FFT beamformer	388984	1794560000	1798528000
	Two-dimensional FFT beamformer	392824	1721500067	
	One-dimensional bitstream FFT beamformer	35395584	43969626736769	44045124208769
	Two-dimensional bitstream FFT beamformer	35399424	41026533245818	

Table 4.6: Beamformer implementation resources comparison. It is a assumed that the decimation filter *multi_0* is used in all beamformers, interpolation rate I=10, interpolation filter length $N_I = 30$, FFT number of points $D_S = 64$ and $N = M = 40$.

In Table 4.6 is shown the required resources to implement a beamformer with specifications as Table 1.1 and Table 1.2, as many decimation filter structures are possible, it is selected the more efficient one (*multi_0*) calculated in Chapter 3 and listed in Table 3.4. As the previous study [8] also has shown, the DAS beamformer implementations in frequency domain mentioned in this chapter are very expensive because of the large memory and computation requirements to implement the FFT blocks. So if the implementation resources are critical, and no additional frequency domain algorithms will be used for the beamformer, the time domain implementation methods are preferred and, in special, the *discrete-time bitstream beamformer* that is the most efficient. If frequency domain implementation is required, for our specifications, it is shown that the state-of-the-art *one-dimensional* and *two-dimensional FFT beamformers* should be preferred.

In Table 4.7 are also summarized the required implementation resources to implement the state-of-the-art *discrete-time beamformer* with specifications as listed in Table 1.1 and Table 1.2 using decimation filters whose required resources are listed in Tables 3.5 and 3.1, the prefix *pcm_* is used to identify these beamformers. The same

	S^{z}_{bf} (bit)	S^{*}_{bf} (MPS)	S^{+}_{bf} (APS)	S^{o}_{bf} (APS)	f_{cpu} (MHz)	T^{+}_{FPGA} (-)	T^{+}_{lp} (-)
pcm_single_direct	1529520	1165886080000	2331648624000	3497548784000	3497548.78	54650	349755
pcm_single_eff	1529520	6072960000	12144624000	18231664000	18231.66	285	1824
pcm_single_memsav	27360	12145280000	12144624000	24303984000	24303.98	380	2431
pcm_multi_0	210040	1213440000	1211504000	4184944000	4184.94	66	419
pcm_multi_1	238280	770560000	766064000	4435184000	4435.18	70	444
pcm_multi_2	318160	160000000	785264000	4451824000	4451.82	70	446
pcm_multi_3	318160	160000000	785264000	4451824000	4451.82	70	446
pcm_multi_4	298600	140160000	1113584000	4558704000	4558.70	72	456
pcm_multi_5	242000	616960000	607344000	4709104000	4709.10	74	471
pcm_multi_6	266360	794240000	789744000	5064304000	5064.30	80	507
pcm_multi_7	243280	389120000	576624000	5067504000	5067.50	80	507
pcm_multi_8	253200	631680000	622064000	5093744000	5093.74	80	510
pcm_multi_9	241120	640000000	632944000	5148144000	5148.14	81	515

Table 4.7: Comparison of required resources to implement a *discrete-time beamformer* using decimation filters listed in Tables 3.4 and 3.1 and beamformer specifications as Table 1.2 (40 microphones).

	S^{z}_{bf} (bit)	S^{*}_{bf} (MPS)	S^{+}_{bf} (APS)	S^{o}_{bf} (APS)	f_{cpu} (MHz)	T^{+}_{FPGA} (-)	T^{+}_{lp} (-)
pdm_single_direct	115310	29270016000	58411008000	90384384000	90384.38	1413	9039
pdm_single_eff	115310	274688000	423408000	3401456000	3401.46	54	341
pdm_single_memsav	77756	426496000	423408000	3553264000	3553.26	56	356
pdm_multi_0	82323	153200000	150080000	3050288000	3050.29	48	306
pdm_multi_1	83029	142128000	138944000	3056544000	3056.54	48	306
pdm_multi_2	85026	126864000	139424000	3056960000	3056.96	48	306
pdm_multi_3	85026	126864000	139424000	3056960000	3056.96	48	306
pdm_multi_4	84537	126368000	147632000	3059632000	3059.63	48	306
pdm_multi_5	83122	138288000	134976000	3063392000	3063.39	48	307
pdm_multi_6	83731	142720000	139536000	3072272000	3072.27	49	308
pdm_multi_7	83154	132592000	134208000	3072352000	3072.35	49	308
pdm_multi_8	83402	138656000	135344000	3073008000	3073.01	49	308
pdm_multi_9	83100	138864000	135616000	3074368000	3074.37	49	308

Table 4.8: Comparison of required resources to implement a *discrete-time bitstream beamformer* using decimation filters listed in Tables 3.4 and 3.1 and beamformer specifications as Table 1.2 (40 microphones).

comparison but for proposal *discrete-time bitstream beamformer* is summarized in Table 4.8, the prefix *pdm_* is used in this case.

It is easy to see that the *discrete-time bitstream beamformer* is the most efficient implementation for all cases at our given conditions except for the *pcm_single_memsav* where the beamformer's storage requirement is less for *discrete-time beamformer* implementation. This difference in storage resources is caused because of the *pcm_single_memsav* decimation filter (Table 3.1) does not meet the (4.27c) condition. In the same way, it could be verified that all others decimation filters are inside (4.27) conditions evaluated with the desired beamformer parameters:

$$S^{+}_{\text{dec}} > 3056000\,,$$

$$S^{*}_{\text{dec}} > 3056000\,,$$

and

$$S^z_{\text{dec}} > 1690\,.$$

Therefore, as these conditions are met, *discrete-time bitstream beamformer* is the most efficient for all those cases.

Finally, after evaluating the implementation resources of the state-of-the-art and proposal methods in time and frequency domains, it is concluded that the proposed *discrete-time bitstream beamformer* is the most efficient for the desired beamformer specification. It is also concluded that, for the same beamformer specification, the frequency domain methods are expensive and not efficient but, in case a frequency domain implementation is required, the state-of-the-art *one-dimensional* or *two-dimensional FFT beamformers* should be prefered.

Chapter 5

Efficient Beamforming

"Given a decimation filter and a beamformer specification, find an architecture that fuses both delay and decimation operations."

In this chapter, in order to meet the above objective formulated in Section 1.2.3 a *delayed decimation filter*, based on Samadi filter, which merges both filtering and delay elements in a single structure is proposed to reduce the resources required in comparison to the already studied structures. This *delayed decimation filter* is equivalent to a decimation filter with low-pass filter impulse response $H(z)$ decimated by R followed by a delay Δ_m, such that the PDM-mic array DAS beamformer shown in Figure 1.1 can be expressed as an array of *delayed decimation filters*.

5.1 Universal maximally flat Samadi filter

The design of linear-phase FIR filters with flat passband and stopband has been first studied by Herrmann [45]. These MAXFLAT filters are characterized by the maximally possible order of tangency at $\omega = 0$ and $\omega = \pi$. Furthermore, their transfer function can be expressed by equivalent closed-form expressions formulated in [45–48]. While Herrmann proposed a design method based on Hermite polynomial interpolation, Rajagpoal [48] derived an equivalent expression based on Bernstein polynomials. In the other hand, nonlinear-phase MAXFLAT FIR filters were introduced by Baher [49]. These filters resulted in improved transition bandwidth and controllable group delay at $\omega = 0$. Even though Baher proposes a simple method to derive MAXFLAT transfer functions, a closed-form was not provided by him.

Finally, Samadi [9] unified Baher's nonsymmetrical filters, linear-phase MAXFLAT filters and Lagrange interpolators using a compact formula for the transfer function of Baher's filters. Samadi's filters can be realized with cellular array structures, as further described in [9, 50].

5.1.1 Samadi's filters

As derived in [9], the transfer function of Samadi's filters is defined by

$$H_{N,K,d}(z) = \sum_{j=0}^{N-K} c_j \Big(\frac{1-z^{-1}}{2}\Big)^{j} \Big(\frac{1+z^{-1}}{2}\Big)^{N-j} \tag{5.1}$$

where

$$c_j = \sum_{i=0}^{j} (-1)^{j-i} \binom{\frac{N}{2}-d}{i} \binom{\frac{N}{2}+d}{j-i}, \tag{5.2}$$

K is the number of zeros at $z=-1$, N is the filter order, and the delay parameter d is a real number defined as

$$d = \alpha - \frac{N}{2}. \tag{5.3}$$

For a given group delay α, such that $0 \leq \alpha \leq N$, one easily verifies that

$$-\frac{N}{2} \leq d \leq \frac{N}{2}$$

or

$$|d| \leq d_{\max} = \frac{N}{2}\,, \tag{5.4}$$

where $d_{\max}$ is the maximum allowed delay parameter and the binomial coefficients in (5.2) are defined as

$$\binom{x}{i} = \begin{cases} \displaystyle\prod_{j=0}^{i-1} \frac{x-j}{j+1}, & i \geq 1 \\ 1, & i = 0 \\ 0. & i < 0 \end{cases} \tag{5.5}$$

This filter becomes a MAXFLAT linear phase FIR when $d = 0$. As shown in [45, 48], the low-pass cutoff frequency (ω_c) of these linear phase filters is related with N as

$$L \simeq \lceil N\omega_c/\pi + 0.5 \rceil \tag{5.6}$$

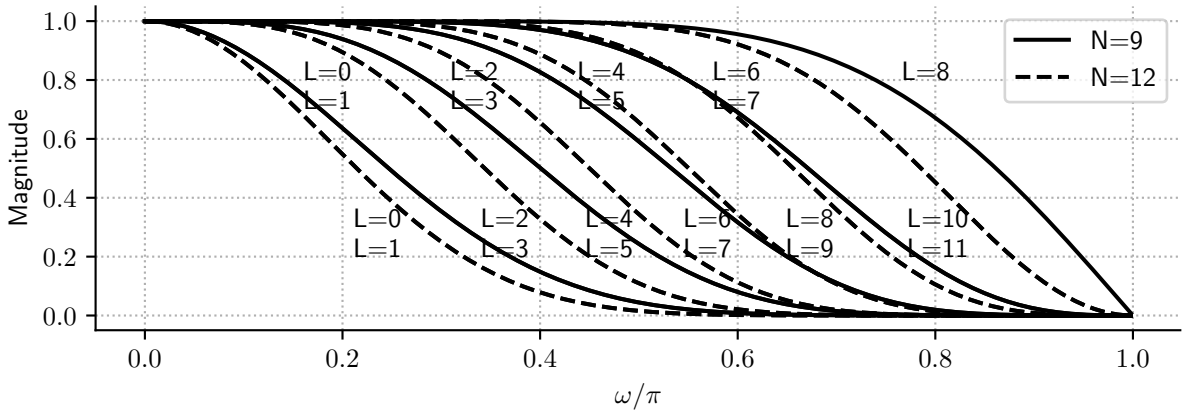
where L is defined for convenience as

$$L = N - K. \tag{5.7}$$

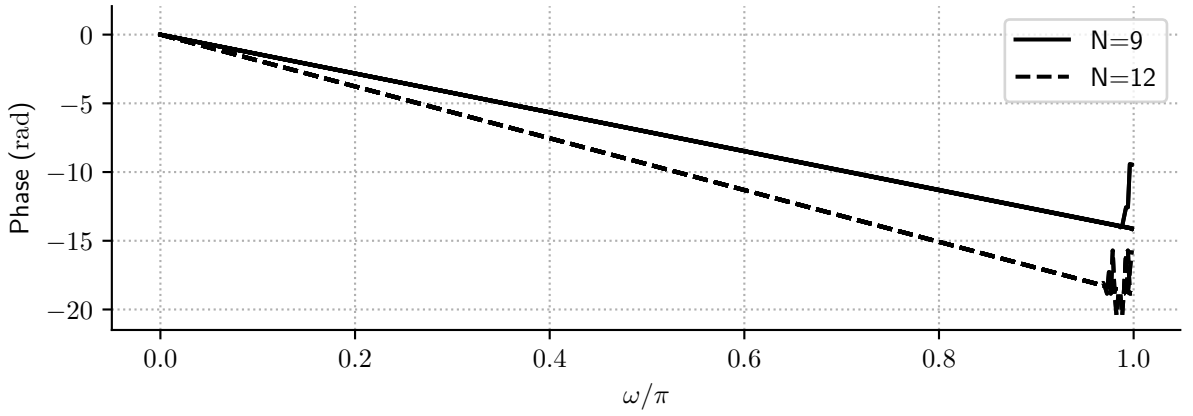
The cutoff frequency of these linear phase filters increases almost linearly with L as shown in Figure 5.1 for different N values. Also, as demonstrated in [9], for linear phase filters ($d = 0$) the coefficient of (5.1) is

$$c_j|_{d=0} = 0, \quad j \text{ odd}\,,$$

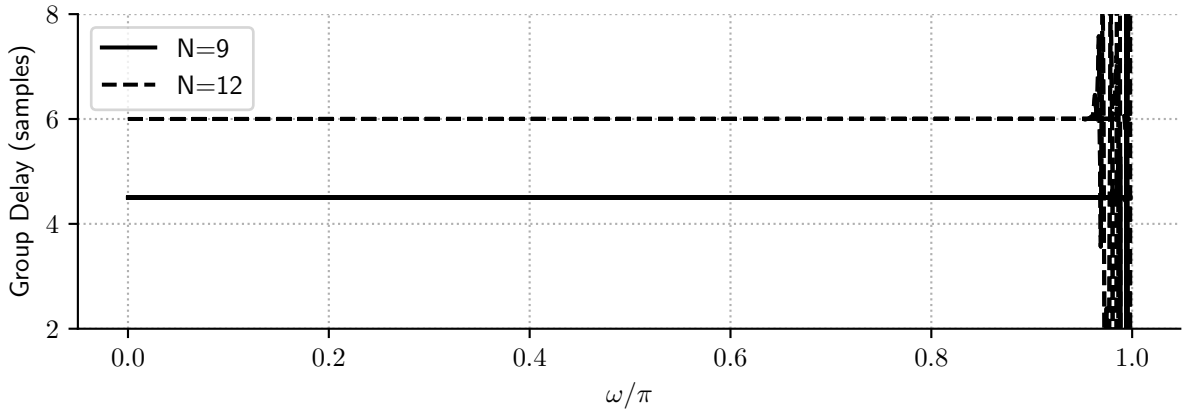
then the magnitude frequency spectrum of $L = 2j$ and $L = 2j + 1$ are the same for $j \in \{0, \ldots, \lfloor N/2 - 1 \rfloor\}$ as shown in Figure 5.1a. Figure 5.1c also shows that the filter group delay for $d = 0$ is $\alpha = N/2$ as expected by (5.3).



(a)



(b)



(c)

Figure 5.1: Linear-phase Samadi'filter magnitude (a), phase (b) and group delay (c) normalized frequency spectrums for $N = 9$ and $N = 12$ ($d = 0$).

In the other hand, when $d \neq 0$ the Samadi filter becomes a MAXFLAT nonlinear phase filter. The most interesting characteristic of this filter class is the ability to modify its group delay as given by (5.3) and shown in Figure 5.2.

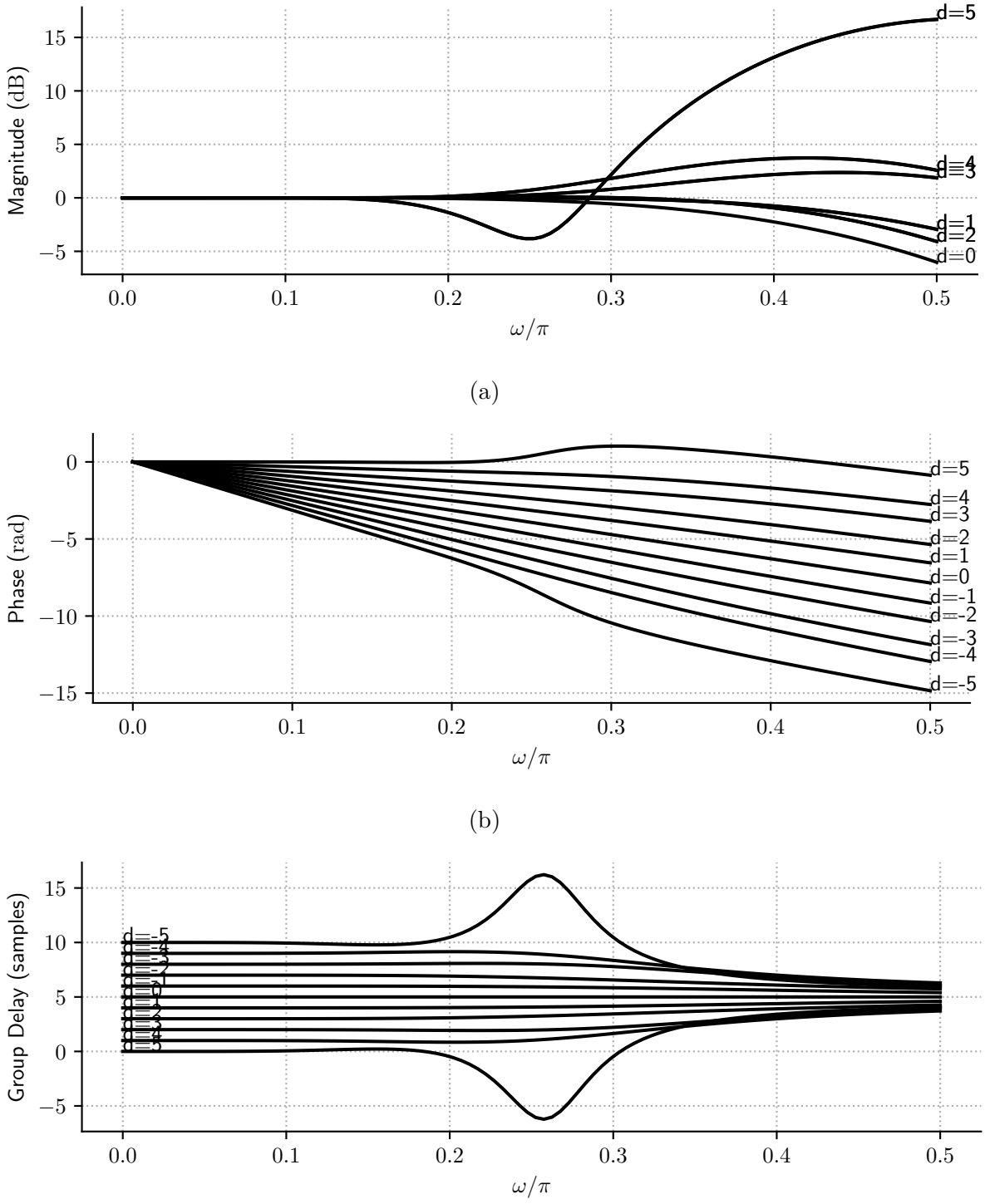


Figure 5.2: Samadi filter magnitude (a), phase (b) and group delay (c) normalized frequency spectrums for $N = 10$ and $d \in \{-5, \ldots, 5\}$

In Figure 5.2 it is shown how the filter flatness is affected when d increases. However, it is also shown that the phase is still linear inside the passband region for $\omega < 0.25\pi$. This suggests that this filter can be used as an intermediary stage in a multirate filter chain to perform group delay adjustment and low-pass filtering as studied in the next section.

5.1.2 Proposal: Samadi filter as multirate filter

A Samadi filter could be used as the LPF of a multirate filter. Given d constant, K and N can be adjusted to meet filter requirements already defined in (2.10). Because Samadi filter frequency spectrum is monotonic, the minimum parameters K and N to meet requirements given by d, R, V_p, V_s, δ_p, δ_s could be calculated with the proposed Algorithm 3, where $R = R_j, \delta_p = \delta_p^j, \delta_s = \delta_s^j, V_p = V_p^j$ and $V_s = V_s^j$, such that if aliasing in transition band is allowed, then $F_p^j = F_p$ for $j \in \{1, \ldots, J-1\}$; otherwise, if aliasing is not allowed, then $F_p^j = F_s$ for $j \in \{1, \ldots, J-1\}$, i.e., the last stage cannot be a Samadi filter.

Figure 5.3 shows examples of minimum N and K values for MAXFLAT linear-phase filters ($d = 0$) using the proposed procedure with decimation factor $R = 2$. As expected, δ_s decreases discretely with higher N values. Also, as pointed out by [51], Figure 5.3b shows how N varies as a quadratic function of ω_p.

Finally, Figure 5.4 shows minimum N and K values calculated by the proposed procedure for $d \in \{0, \ldots, 26\}$ and common values of ω_p. It is shown that the minimum N, required for any d, decreases with ω_p increments, and it is almost 3 times d when $\omega_p/\pi = 0.28$.

5.1.3 Samadi's decimation filter implementation

Binomial chain structure

Since the Samadi filter equation (5.1) is a binomial filter sequence (as first proposed by Haddad in [52]), this filter can be expressed as

$$H_{N,K,d}(z) = \Big(\frac{1+z^{-1}}{2}\Big)^N \sum_{j=0}^{N-K} c_j \Big(\frac{1-z^{-1}}{1+z^{-1}}\Big)^j. \tag{5.8}$$

The binomial equation (5.8) can be realized as a cascade of two filters:

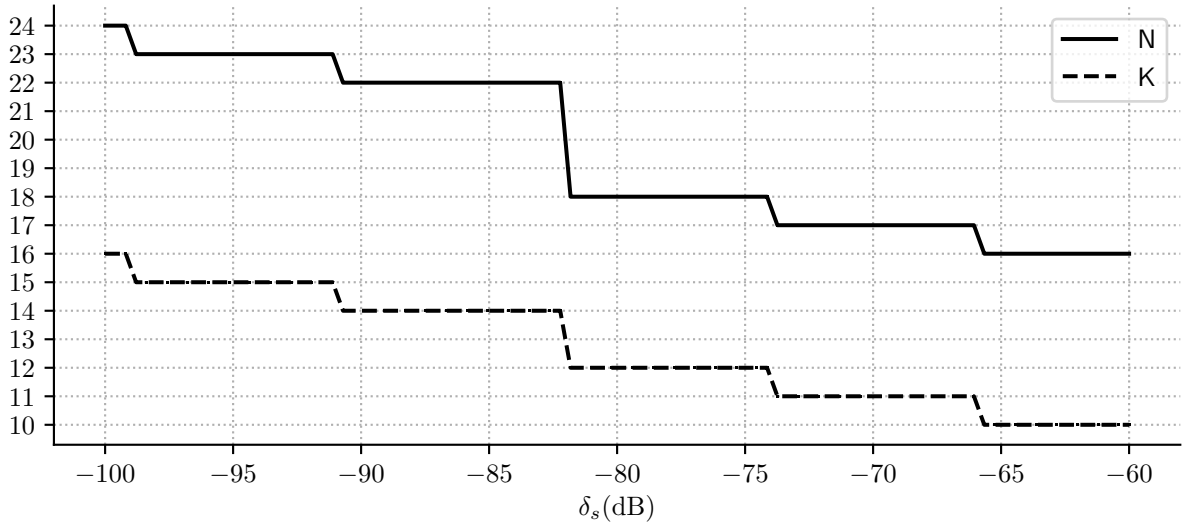
$$H_{N,K,d}(z) = A_N(z) B_{N,K,d}(z) \tag{5.9}$$

where

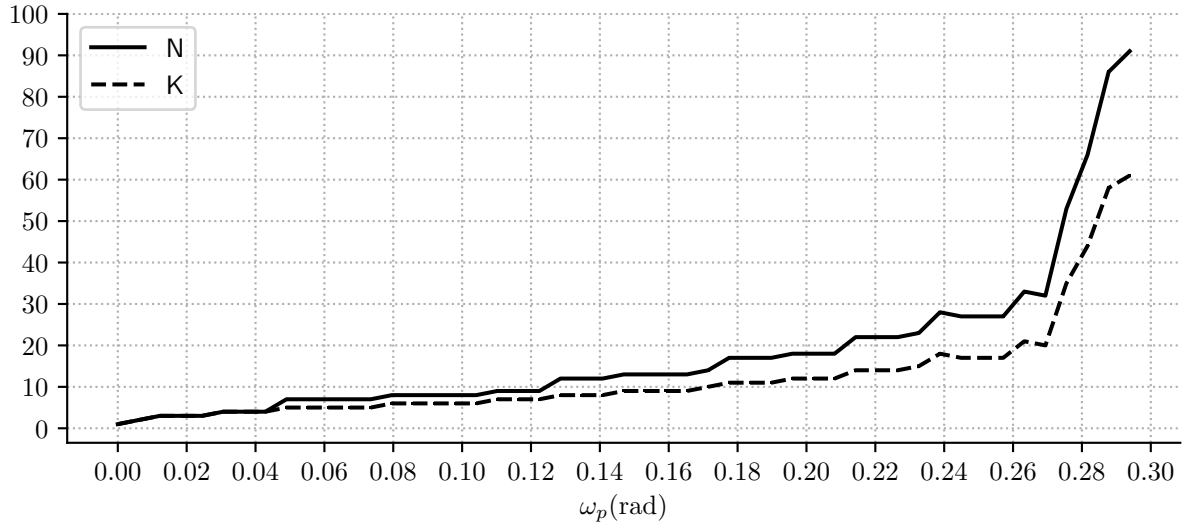
$$A_N(z) = \Big(\frac{1+z^{-1}}{2}\Big)^N, \quad B_{N,K,d}(z) = \sum_{j=0}^{N-K} c_j \Big(\frac{1-z^{-1}}{1+z^{-1}}\Big)^j, \tag{5.10}$$

Algorithm 3 Samadi's Filter minimum N and K calculation algorithm

```
1: procedure SamadiMinN(d, R, δ_p, δ_s, V_p, V_s)
2:     L ← 0
3:     N ← 2⌈d⌉
4:     loop
5:         δ'_p ← max(|H_{N,N−L,d}(e^{iω}) − 1|)   ∀ω ∈ V_p
6:         δ'_s ← max(|H_{N,N−L,d}(e^{iω})|)   ∀ω ∈ V_s
7:         if d = 0 then                                   ▷ Linear-phase filter
8:             if δ'_p ≤ δ_p and δ'_s ≤ δ_s then
9:                 K = N − L
10:                return N, K
11:            else if L ≤ ⌈Nω_p/π + 0.5⌉ then
12:                L ← L + 2
13:            else
14:                L ← 0
15:                N ← N + 1
16:            end if
17:        else                                            ▷ Nonlinear-phase filter
18:            if δ'_p ≤ δ_p and δ'_s ≤ δ_s then
19:                K = N − L
20:                return N, K
21:            else if δ'_s ≥ 1 or L ≥ N then
22:                L ← 0
23:                N ← N + 1
24:            else
25:                L ← L + 1
26:            end if
27:        end if
28:    end loop
29: end procedure
```



(a)



(b)

Figure 5.3: Minimum N and K values for linear-phase Samadi filter ($d = 0$), decimation factor $R = 2$, passband frequency $\omega_p = 2\pi/R - \omega_s$ and passband ripple $\delta_p = 0.1\,\text{dB}$. (a) In function of δ_s, $\omega_p = 0.21$ constant. (b) In function of ω_p, $\delta_s = -80\,\text{dB}$ constant.

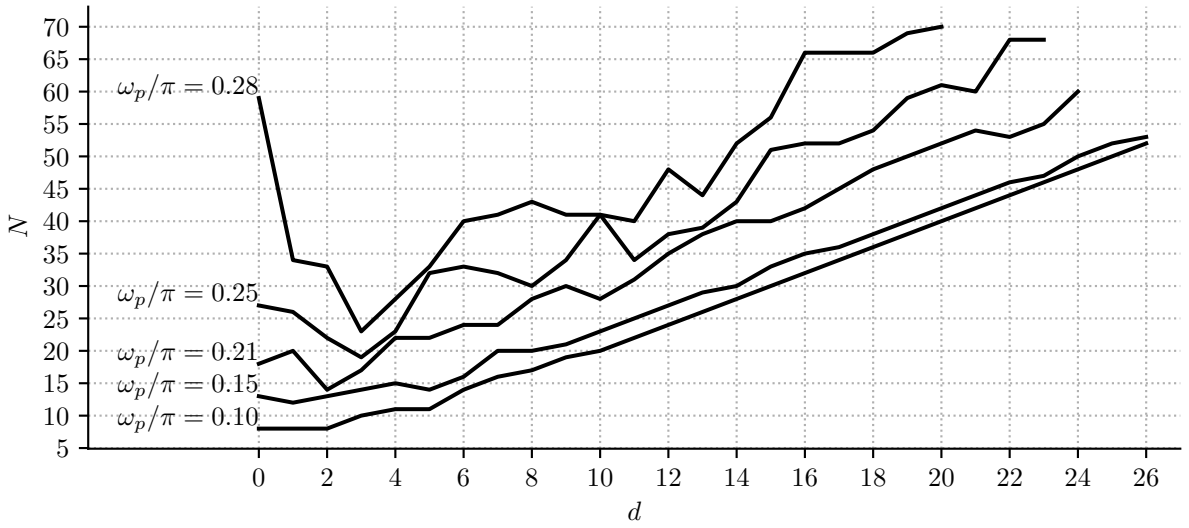
as depicted in Figure 5.5a and 5.5b.

We will call the block diagram in Figure 5.5a as binomial representation Type I, which requires the following resources

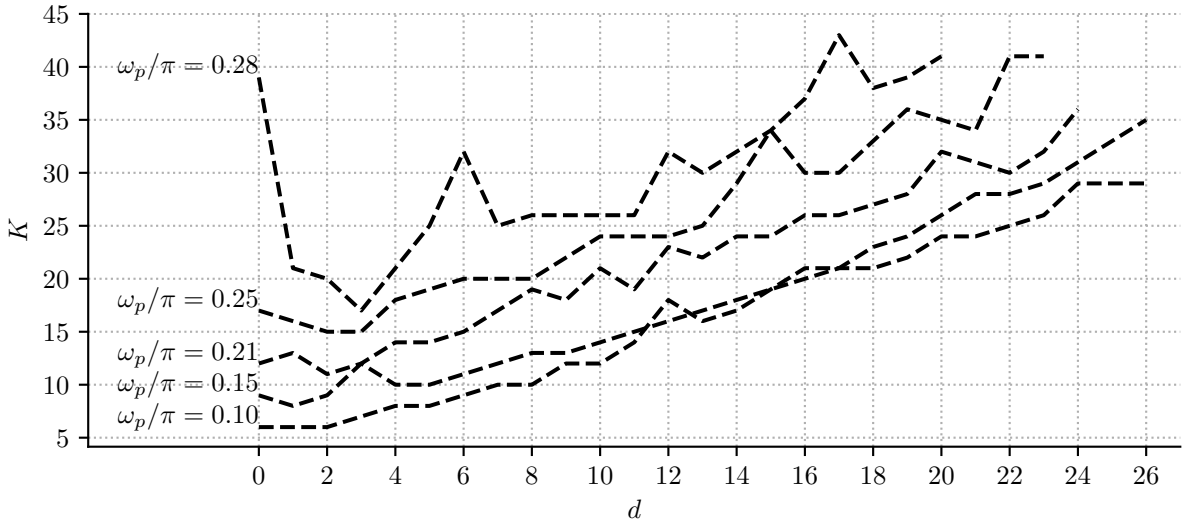
$$S^z_{\text{dec}} \simeq (2N - K)L_{\text{acc}} \text{ bit} \tag{5.11a}$$

$$S^*_{\text{dec}} = (N - K)f_o \tag{5.11b}$$

$$S^+_{\text{dec}} = (5N - 4K)f_o R, \tag{5.11c}$$



(a)



(b)

Figure 5.4: Minimum N (a) and K (b) values depending on d and required ω_p for $R = 2$ and $\delta_s = -80\,\mathrm{dB}$.

and we call Figure 5.5b as binomial representation Type II, which requires the following resources

$$S^z_{\mathrm{dec}} = S^z_{\mathrm{dec}A} + S^z_{\mathrm{dec}B} \tag{5.12a}$$

$$S^*_{\mathrm{dec}} = S^*_{\mathrm{dec}A} + S^*_{\mathrm{dec}B} \tag{5.12b}$$

$$S^+_{\mathrm{dec}} = S^+_{\mathrm{dec}A} + S^+_{\mathrm{dec}B}, \tag{5.12c}$$

where

$$S^{z}_{\mathrm{dec}A} \simeq NL_{\mathrm{acc}} \text{ bit} \tag{5.13a}$$
$$S^{*}_{\mathrm{dec}A} = 0 \tag{5.13b}$$
$$S^{+}_{\mathrm{dec}A} = Nf_oR. \tag{5.13c}$$

and

$$S^{z}_{\mathrm{dec}B} \simeq (N-K)L_{\mathrm{in}} \text{ bit} \tag{5.14a}$$
$$S^{*}_{\mathrm{dec}B} = (N-K)f_oR \tag{5.14b}$$
$$S^{+}_{\mathrm{dec}B} = 4(N-K)f_oR. \tag{5.14c}$$

are the required resources to implement $A_N(z)$ and $B_{N,K,d}(z)$, respectively.

It easy to see that the Type I implementation requires less multiplications per second (S^{*}_{dec}) than Type II implementation, as they are performed after decimation. Even though Type II requires more resources, as the N blocks at the end do not depend on c_j coefficients, its usage could be more convenient in some PAPS applications when sharing resources is allowed, such that the N blocks are shared between channels. This Type II implementation sharing resources property will be used in Section 5.3 to reduce required resources in beamforming implementation.

Celular structure

As further detailed by Samadi in [50], the filter $H_{N,K,d}(z)$ can be implemented as a cellular structure, like the structure presented in Figure 5.6. Defining $P_{i,j}$ as a node localized at row i and column j in the cellular structure, one can verify that those nodes are related to each other according to

$$P_{i,j} = \begin{cases} c'_j x[n], & i = 0, \\ D(z)\begin{bmatrix} P_{i-1,j} \\ P_{i-1,j+1} \end{bmatrix}, & \text{otherwise}, \end{cases} \tag{5.15}$$

where, recalling that c_j is defined by (5.2), we have that

$$c'_j = \begin{cases} 1, & j = 0, \\ \dfrac{c_j}{\binom{N}{j}}, & 0 < j \leq N-K, \\ 0, & \text{otherwise}, \end{cases} \tag{5.16}$$

$$D(z) = \frac{1}{2}\begin{bmatrix} 1+z^{-1} & 1-z^{-1} \end{bmatrix}, \tag{5.17}$$

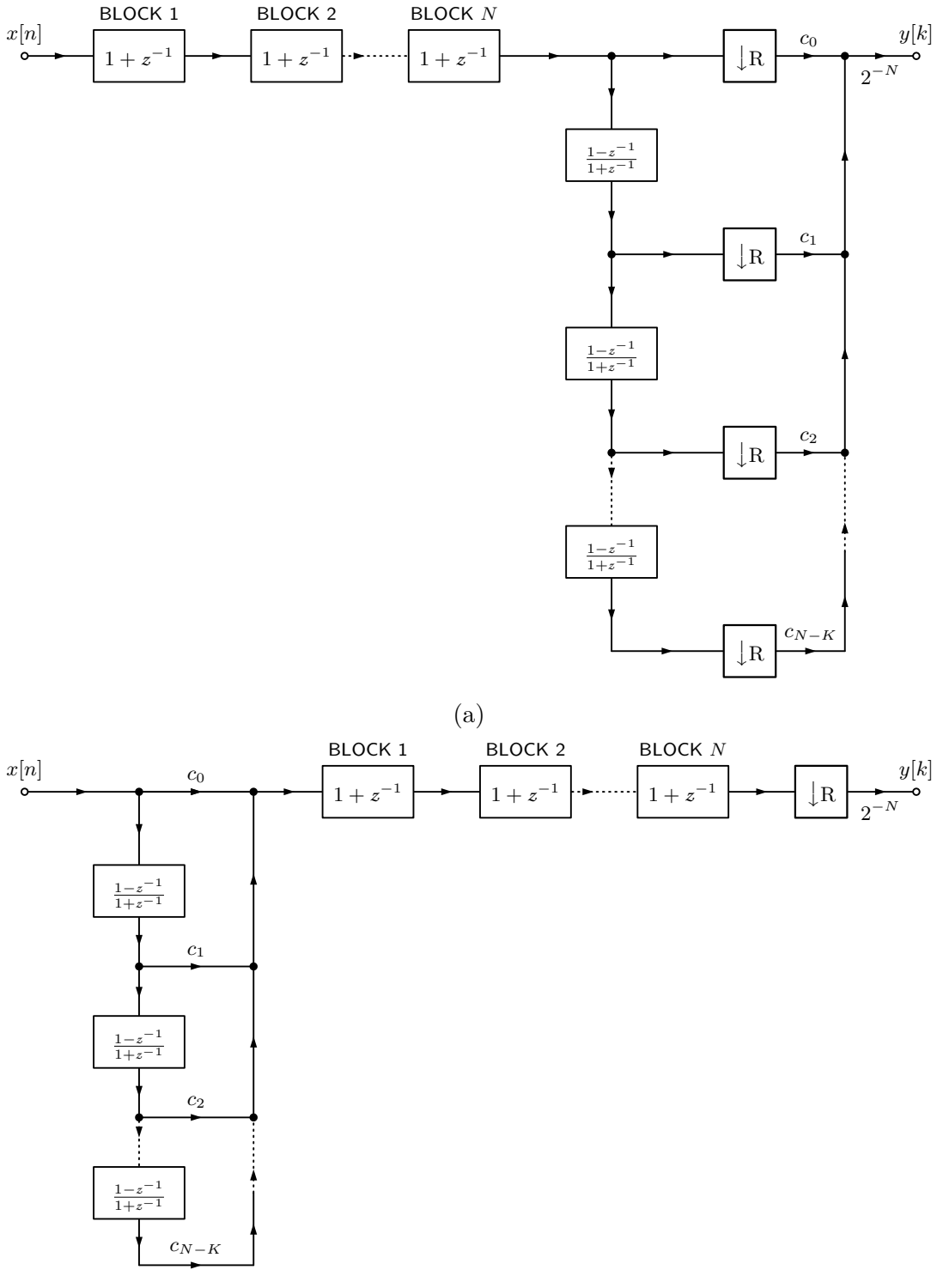


Figure 5.5: Samadi's decimation filter binomial representation Type I (a) and Type II (b).

and the filter output is given by

$$y[k] = P_{N-K+1,0}. \tag{5.18}$$

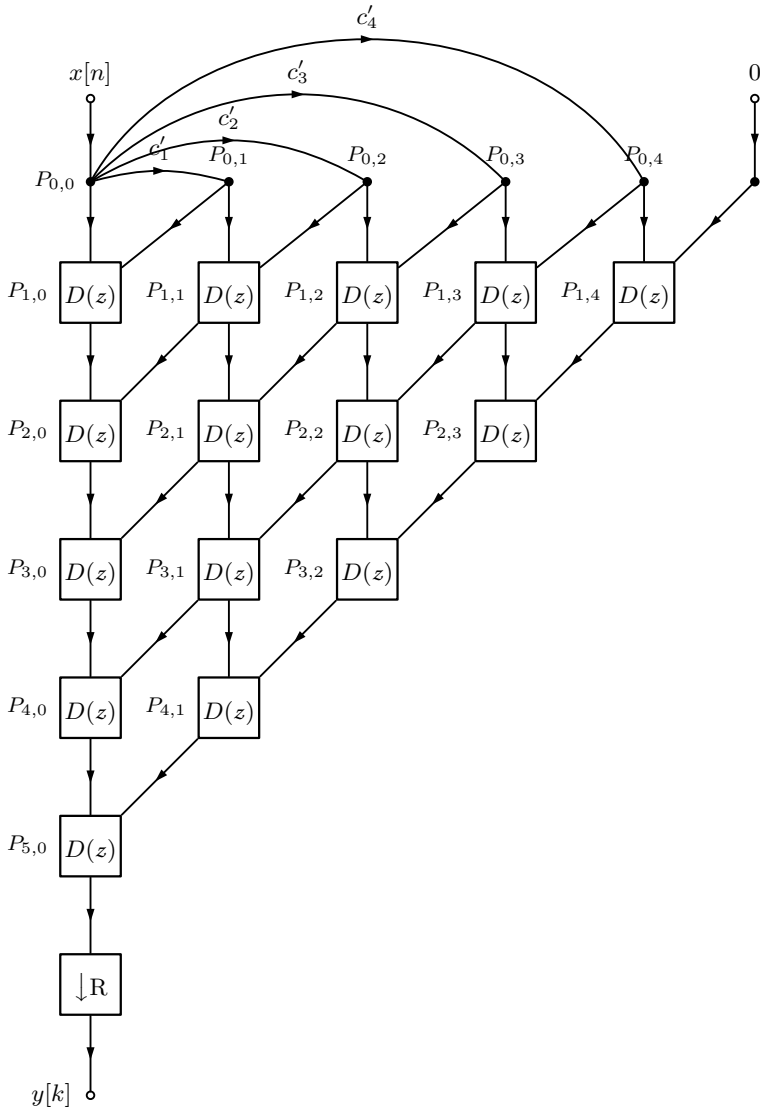


Figure 5.6: Samadi's decimation filter cellular structure example for $N = 5$ and $K = 1$.

Finally, the resources required to implement a jth-stage $H_{N,K,d}(z)$ filter with this structure are:

$$S^z_{\text{dec}} \simeq \frac{1}{2}(N-K)(N-K+1)L_{\text{acc}} \text{ bit} \tag{5.19a}$$

$$S^*_{\text{dec}} = (N-K)f_oR \tag{5.19b}$$

$$S^+_{\text{dec}} = \frac{3}{2}(N-K)(N-K+1)f_oR \tag{5.19c}$$

Even though this implementation requires more resources than binomial implementations, it is specially suitable for configurable filter realizations where its parameters can be varied by adding or deleting extra cells or changing the value of a single c_j coefficient. For instance, a filter that requires the cutoff frequency ω_p be changing dynamically would

only need to keep zeroing c'_j coefficients according to the respective K value, without any further change in the filter structure.

5.2 Proposal: Delayed Decimation Filter

As one objective of this chapter is to find a decimation filter that fuses both delay and decimation to then be used in a beamformer and as the group delay is the parameter that represents the filter input to output delay, we need to find a decimation filter architecture with configurable group delay (Figure 5.7a).

As shown in previous sections, Samadi filter is the best candidate for this purpose as its group delay could be regulated just changing the Samadi filter delay parameter (d) and, as d is a real number, this group delay is not restricted to discrete values only. Samadi's filters have also the advantage to have a flat passband response (MAXFLAT), but they have the disadvantage that their passband δ_p and stopband δ_s ripples worsen as their Samadi filter delay parameter (d) increases (see Figure 5.2a).

As the Samadi filter design depends only on three parameters N, K and d, if the two first ones are kept constant and only d is variable, the δ_s and δ_p parameters will change along with this parameter. So if it is defined a maximum d ($d_{\max}$) such that $|d| \leq d_{\max}$, it is necessary that the Samadi filter will be designed to keep its δ_s and δ_p parameters under required specification for all d values allowed.

Also, as a Samadi filter does not have the flexibility to be designed for specific F_p and F_s values without changing other filter parameters, its frequency response needs to be compensated to keep the overall decimation filter's parameters under specification.

So, due to these mentioned Samadi filter limitations, in order to keep the overall decimation filter's parameters under specification, this thesis proposes a J-stages decimation filter whose penultimate stage ($J-1$) is a Samadi filter and its last stage (J) is an equiripple filter as shown in Figure 5.7b. The Samadi filter will control the whole filter group delay by setting its respective d parameter and the last equiripple stage will compensates the magnitude and phase distortion caused by Samadi filter for all $|d| \leq d_{\max}$. The Samadi filter could be implemented either as a cellular structure or as a binomial chain structure (Figure 5.7c) already discussed in previous sections. Also, as this is a multi-stage filter, other filtering stages (1 to $J-2$) can be added before the Samadi filter to help in the decimation.

In the end, the proposed *delayed decimation filter* will be an "all-in-one" filter that performs the same filtering and downsampling operations that any state-of-the-art decimation filter and has the capability of alter its group delays without any change on its structure or additional delay chain.

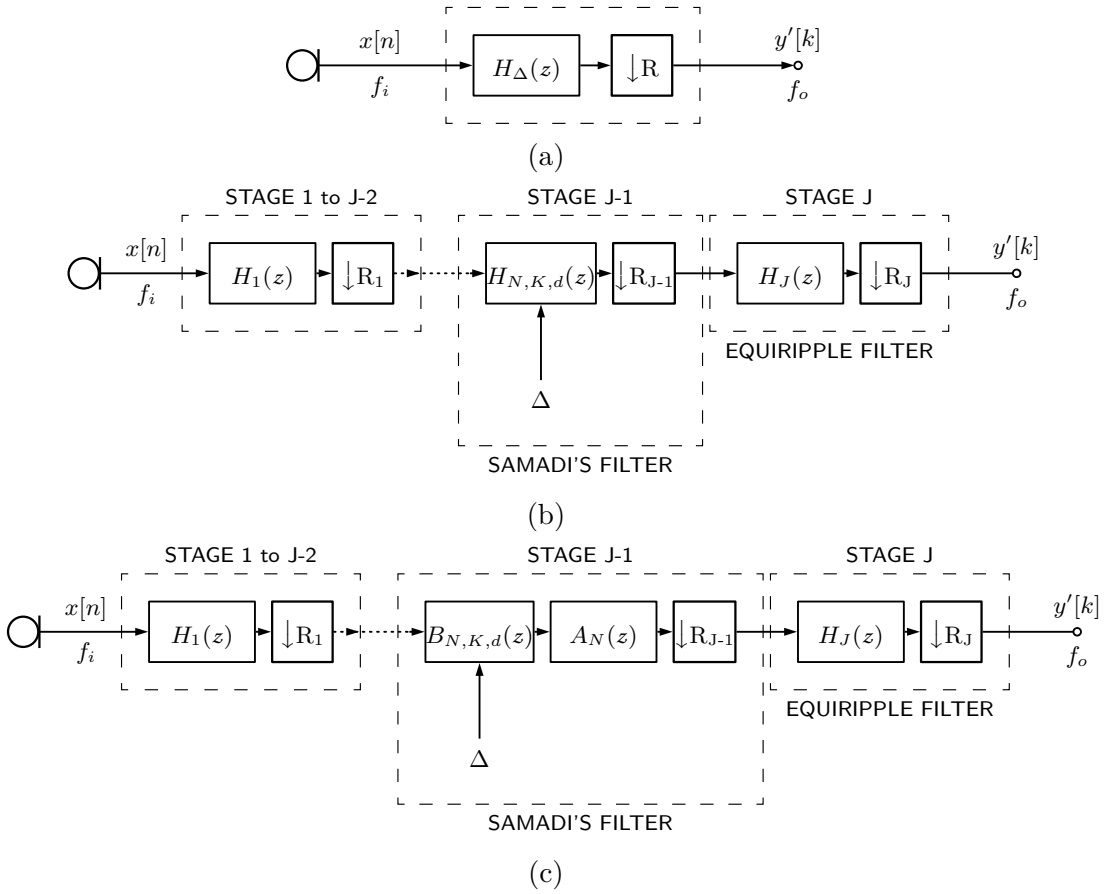


Figure 5.7: (a) *Delayed decimation filter*, (b) its version as a multi-stage decimation filter with $J-1$ stage being a Samadi filter, (c) and its version with Samadi filter decomposed in its binomial components.

5.2.1 Design considerations

As discussed in previous sections, the three design parameters of a Samadi filter are the number of zeros at $z=-1$ in a Samadi filter (K), the Samadi filter order (N) and the Samadi filter delay parameter (d). In this case, as the d parameter will vary in the range $|d| \leq d_{\max}$, $d_{\max}$ will be our design parameter instead of d.

The filter delay Δ depends on the d parameter, R_{J-1} and R_J parameters as follows:

$$\Delta = \frac{d}{R_J R_{J-1} f_o}. \tag{5.20}$$

Therefore, if $|d| \leq d_{\max}$ then

$$|\Delta| \leq \frac{d_{\max}}{R_J R_{J-1} f_o} \quad \text{or} \quad \Delta_{\max} = \frac{d_{\max}}{R_J R_{J-1} f_o}. \tag{5.21}$$

This expression says that maximum required delay ($\Delta_{\max}$) is limited by the $d_{\max}$ parameter. So, for f_o and $\Delta_{\max}$ specifications as in Tables 1.1 and 1.2 respectively, and

assuming that $R_J = R_{J-1} = 2$, by (5.21), $d_{\max} = 20.13$. Then, the parameters N and K need to be calculated in order that overall filter specification will be kept for all $|d| \leq 20.13$.

Once $d_{\max}$ is defined, the minimum K and N parameters can be calculated using Algorithm 3 with the desired filter specification and $d_{\max}$ instead of d as inputs. It is important to remark that if the Samadi filter is designed for $d_{\max}$ the decimation filter will continue under the same specification for values $|d| \leq d_{\max}$, this effect can be observed in Figure 5.2a, where δ_p decreases for lower values of d, and in Figure 5.4, where for $d \geq 3$ given a combination (d, N, ω_p) if N is kept constant and d is decreased, ω_p will always increase, so the flatness will be improved.

5.2.2 Physical limit of the d parameter

Given a decimation filter specification, $d_{\lim}$ is defined as the physical limit of the d parameter, such that

$$|d| \leq d_{\max} \leq d_{\lim} ,$$

such that if any value of $|d|$ is above this limit, independent of the group delay, it would cause a violation in the desired decimation filter specification. Figure 5.8 shows the $d_{\lim}$ values for a decimation filter with specifications as listed in Tables 1.1 but with F_p varying in the range from 0 kHz to 7.5kHz range. The $d_{\lim}$ value in this figure was calculated, for each F_p frequency, designing first the *delayed decimation filter* with Algorithms 2 and 3, and then increasing d by 1; this process is repeated until d reaches the limit value $d_{\lim}$ that still keeps the filter under desired specification; N and K are calculated for each $d_{\lim}$.

Figure 5.8 shows that the physical limit of the d parameter is approximately $d_{\lim} \approx 25$ around the Nyquist's frequency and around $d_{\lim} \approx 50$ for low frequencies and that $d_{\lim}$ decreases as F_p increases. This means that a filter with the same specification but $d_{\max} = 30$ and $F_p = 7.5$kHz will not be realizable with the proposed *delayed decimation filter* because $d_{\lim} \approx 25$ at this F_p point.

So, the $d_{\lim}$ value should be taken into account at design time as this will limit the maximum allowed delay parameter ($d_{\max}$) and consequently the maximum delay of the decimation filter ($\Delta_{\max}$).

5.2.3 Implementation resources

Assuming that S_j^z, S_j^+, S_j^*, and S_j^o are the storage requirements, number of additions per second, number of multiplications per second and total number of additions per second respectively of the jth-stage of the *delayed decimation filter* for $j = 1, \ldots, J$; it is easy to see from Figure 5.7c that the respective implementation resources for the

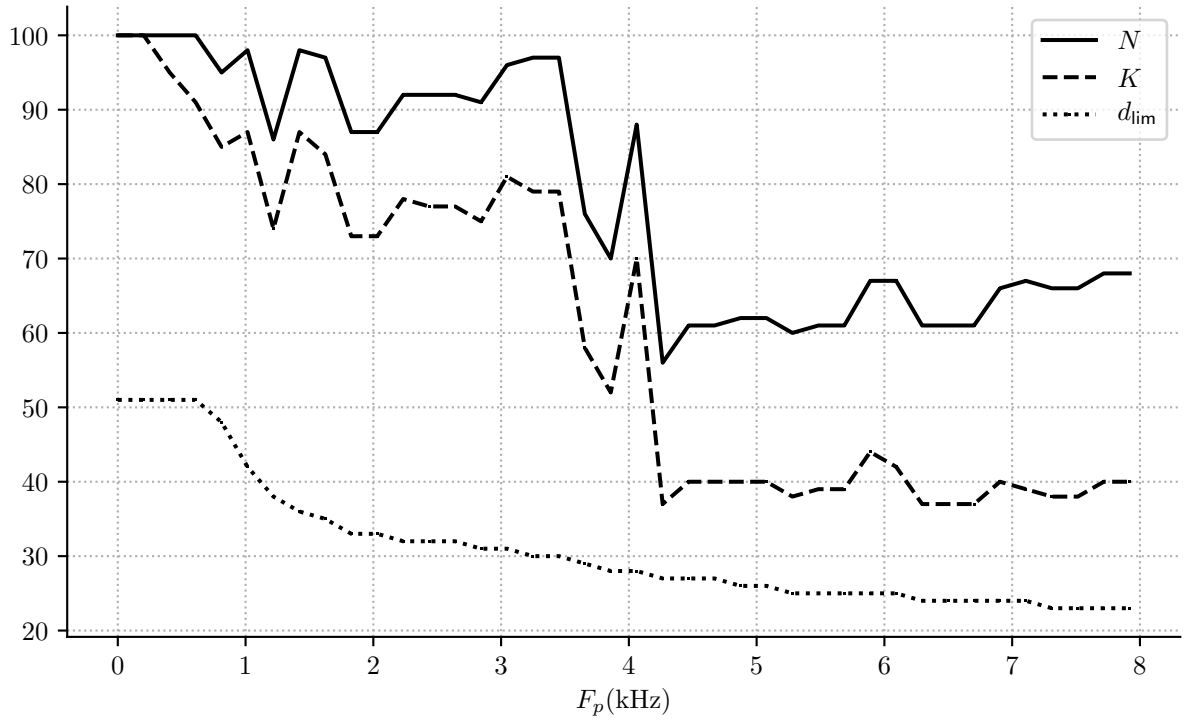


Figure 5.8: d_{lim} for a 3-stages *delayed decimation filter* with structure ['lthband', 'maxflat', 'equir'], decimation rates [48, 2, 2] and filter requirements specified in Table 1.1 but F_p variable; The N and K values correspond to the respective d_{lim} values.

delayed decimation filter will be

$$S^z_{\text{dec}} = \sum_{j=1}^{J} S^z_j = \sum_{j=1}^{J-2} S^z_j + S^z_{\text{dec}A} + S^z_{\text{dec}B} + S^z_J \,, \tag{5.22a}$$

$$S^+_{\text{dec}} = \sum_{j=1}^{J} S^+_j = \sum_{j=1}^{J-2} S^+_j + S^+_{\text{dec}A} + S^+_{\text{dec}B} + S^+_J \,, \tag{5.22b}$$

$$S^*_{\text{dec}} = \sum_{j=1}^{J} S^*_j = \sum_{j=1}^{J-2} S^*_j + S^*_{\text{dec}A} + S^*_{\text{dec}B} + S^*_J \,, \tag{5.22c}$$

$$S^o_{\text{dec}} = \sum_{j=1}^{J} S^o_j = \sum_{j=1}^{J-2} S^o_j + S^o_{\text{dec}A} + S^o_{\text{dec}B} + S^o_J \,, \tag{5.22d}$$

where $S^z_{\text{dec}A}$ and $S^z_{\text{dec}B}$ are the storage requirements for the $A_N(z)$ and $B_{N,K,d}(z)$ filters respectively, the same for the other resource variables.

5.3 Proposal: Beamformer based on delayed decimation Filter

The Samadi filter stage in a *delayed decimation filter* in Figure 5.7c can be expressed in its binomial representation Type II (Figure 5.5b), in such way that the latter part of the filter chain does not depend on Δ. Therefore, if M *delayed decimation filters* are placed in parallel, the weightings by w_m are placed just before the $A_N(z)$ filter and their outputs are added to form a beamformer, as shown in Figure 1.3, the latter part after $B_{N,K,d}(z)$ could be shared between all microphone channels as shown in Figure 5.10.

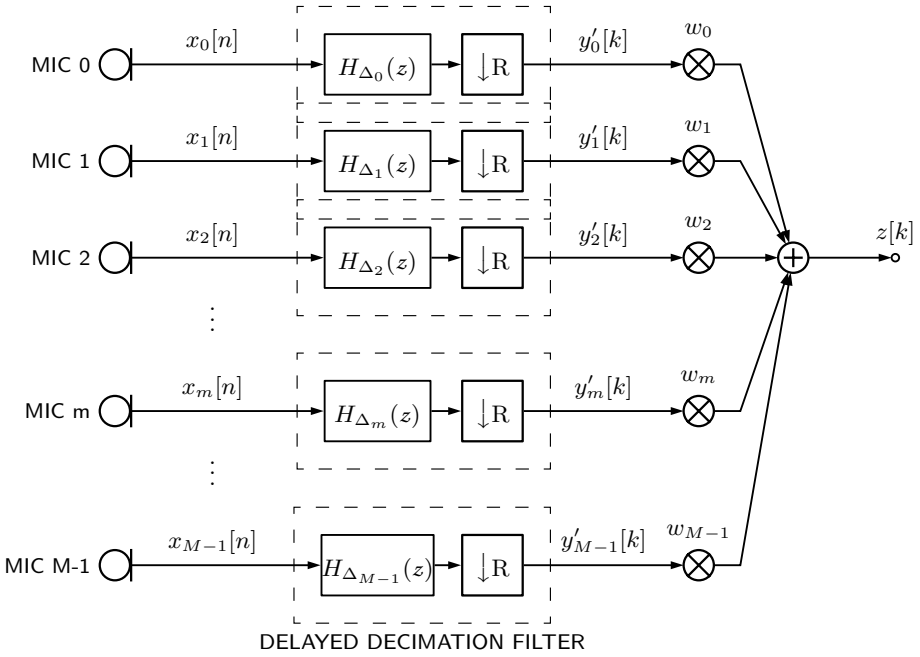


Figure 1.3: PDM-mic array DAS beamformer using *delayed decimation filter*s (repeated from page 26)

Note also, in the same way that beamformers at PDM domain discussed in Chapter 4, the beamforming is not performed with the final PCM signals at the output sampling rate (f_o), rather the beamforming is performed in an intermediary stage of the decimation filter at the $R_J R_{J-1} f_o$ rate.

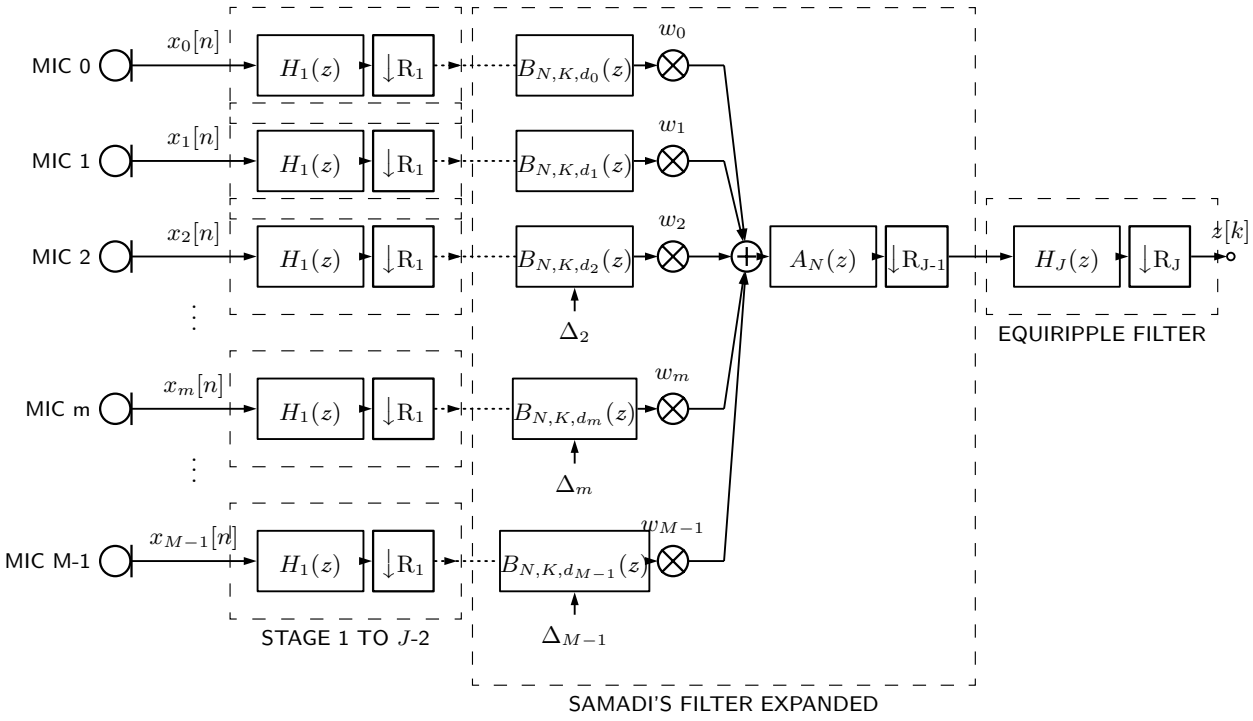


Figure 5.10: PDM-mic array DAS beamformer using *delayed decimation filter*s

Finally, the required resources to implement this beamformer can be calculated from (5.23) and Figure 5.10 as following:

$$S^z_{\text{bf}} = M\sum_{j=1}^{J-2} S^z_j + MS^z_{\text{dec}B} + S^z_{\text{dec}A} + S^z_J\,, \tag{5.23a}$$

$$S^+_{\text{bf}} = M\sum_{j=1}^{J-2} S^+_j + MS^+_{\text{dec}B} + S^+_{\text{dec}A} + S^+_J + (M-1)R_JR_{J-1}f_o\,, \tag{5.23b}$$

$$S^*_{\text{bf}} = M\sum_{j=1}^{J-2} S^*_j + MS^*_{\text{dec}B} + S^*_{\text{dec}A} + S^*_J + MR_JR_{J-1}f_o, \tag{5.23c}$$

$$S^o_{\text{bf}} = M\sum_{j=1}^{J-2} S^o_j + MS^o_{\text{dec}B} + S^o_{\text{dec}A} + S^o_J + (M-1+(L_{\text{out}}-1)M)R_JR_{J-1}f_o\,, \tag{5.23d}$$

where the last term summed to S^+_{bf}, S^*_{bf} and S^o_{bf} are the respective additional required resources to implement the adder and the weighting w_m blocks, these calculations being at the $R_JR_{J-1}f_o$ sampling rate.

5.4 Results

A *delayed decimation filter* was designed according to specifications listed in Table 1.1 using Algorithms 2 and 3, the filter has 3-stage architecture (['lthband', 'maxflat',

'equir']) with decimation rates: [48, 2, 2], as shown in Figure 5.11. Figure 5.12 shows the individual frequency spectrum of each internal stage for $d_{\max} = 20.13$. Note that even though the *maxflat* stage has a bumpy frequency spectrum above the passband frequency (F_p), this is compensated by the last stage equiripple filter (*equir*) as shown in Figure 5.13.

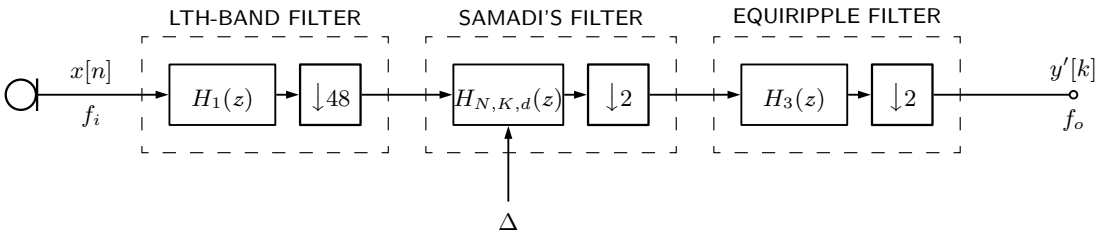
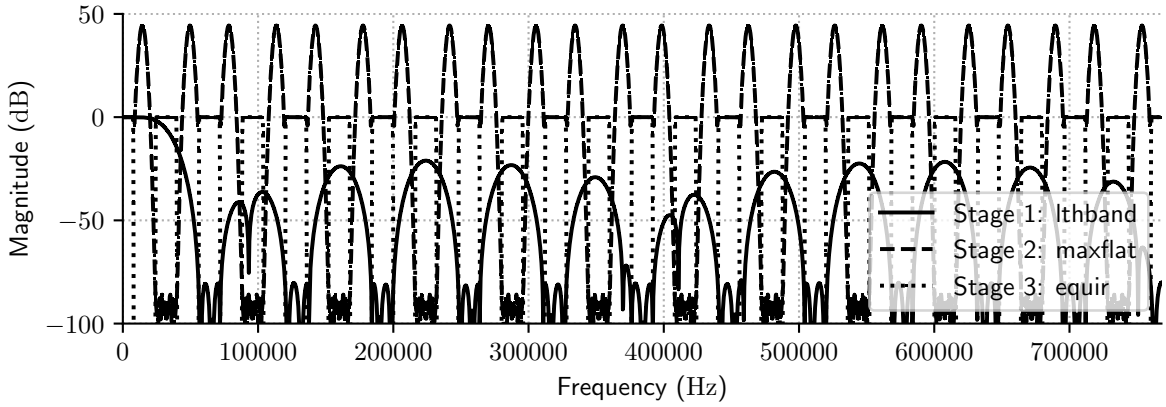


Figure 5.11: *Delayed decimation filter* for a PDM-mic

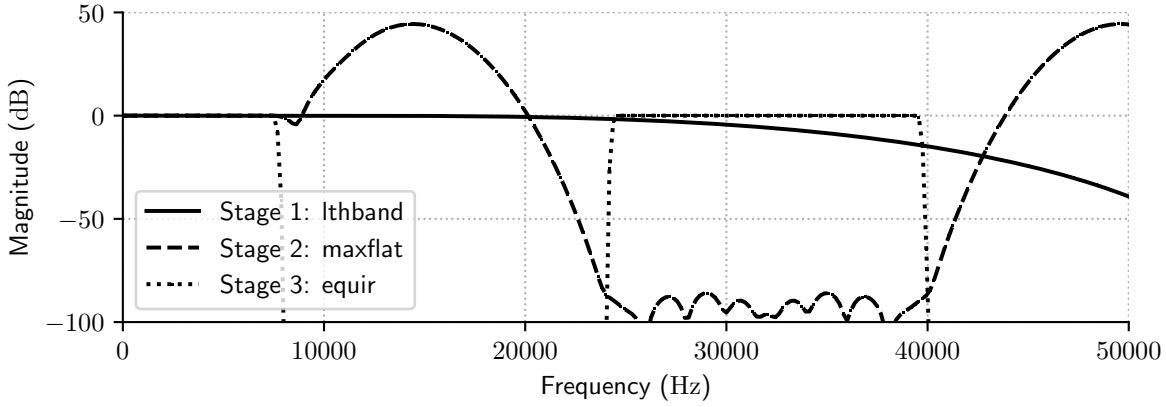
As mentioned along this chapter, the advantage of using a Samadi filter is that it allows one to change its group delay just by changing some coefficients, i.e., without any change in the whole filter structure. Figure 5.14 shows the group delay of this multi-stage filter for many values of its d parameter. It is easy to see how the group delay is directly proportional to the d parameter.

Also, the breakdown of the resource requirements by stage is shown in Table 5.2, where the first row corresponds to the Lth-band filter stage, the second and third rows correspond to the $B_{N,K,d}(z)$ and $A_n(z)$ parts of the Samadi filter, respectively, and the last row corresponds to the equiripple filter. The total resource requirements to implement this filter are shown in Table 5.4 as *delayed*along with the resource requirements of multi-stage filter architetures found in Chapter 3 by optimization algorithm. It is noted that this standalone decimation filter is not more efficient than the filters found by the optimization algorithm, it requires in general more storage and computational resources for its implementation. Even though this proposed *delayed decimation filter* requires more implementation resources, it offers the additional capability to regulate its group delay that could be an advantage in some applications like beamforming.

Finally, Table 5.3 shows the resources required to implement a DAS beamformer based on this 3-stages *delayed decimation filter* designed for array specifications listed in Table 1.2. This result is compared to other beamformer implementations discussed in Chapter 4 in Table 5.4. It is observed that the proposed architecture requires less computational resources than the other beamformers and that, after the *pmc_single_memsav* beamformer, the proposed architecture requires also less storage. However, as the *pmc_single_memsav* beamformer is the architecture that also requires a prohibite quantity of computational resources, it can be concluded that the proposed beamformer based on *delayed decimation filters* is the more resource-efficient beamformer architecture for the given specification.



(a)

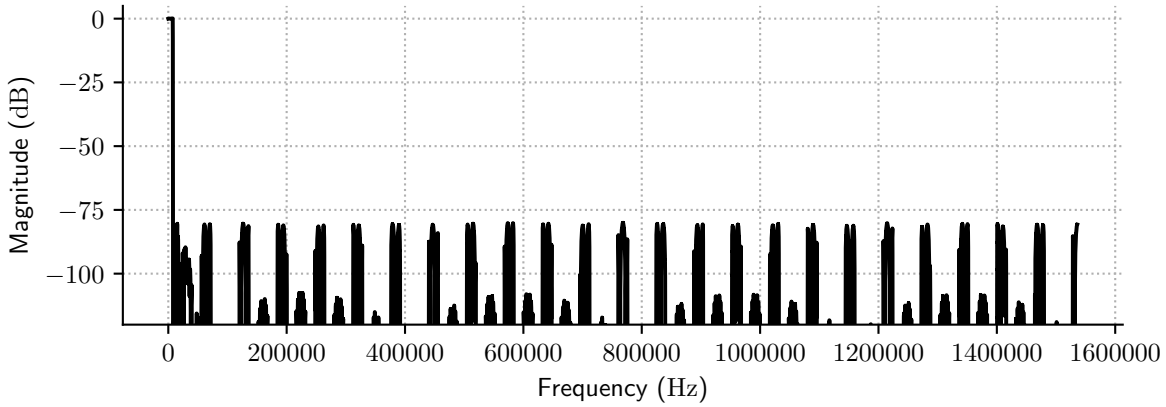


(b)

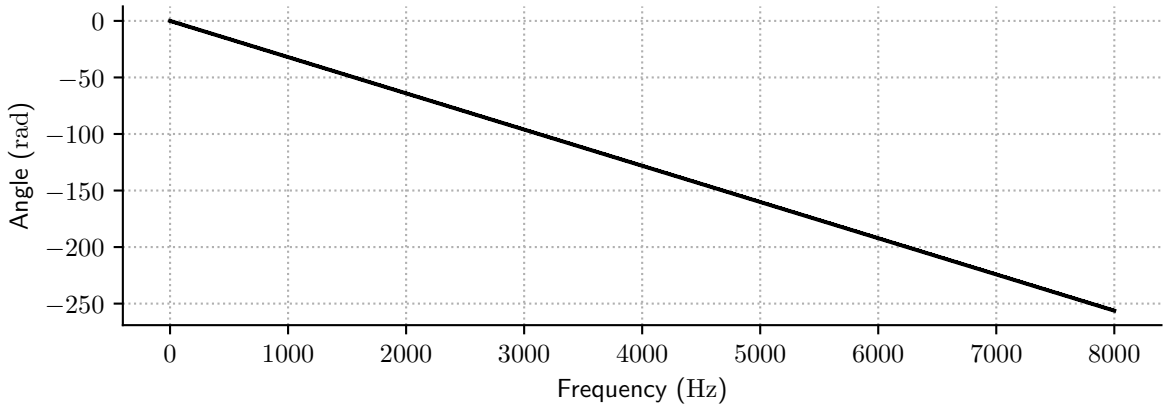
Figure 5.12: (a) Magnitude frequency spectrum of internal stages of the *delayed decimation filter* in the whole input range, and (b) the same frequency spectrum in the 0 kHz to 50 kHz range.

	S^z_{dec} (bit)	S^*_{dec} (MPS)	S^+_{dec} (APS)	S^o_{dec} (APS)	f_{cpu} (MHz)	T^+_{FPGA} (-)	T^+_{lp} (-)
multi_0	4963	30320000	30272000	104240000	104.24	2	11
multi_1	5669	19248000	19136000	110496000	110.50	2	12
multi_2	7666	3984000	19616000	110912000	110.91	2	12
multi_3	7666	3984000	19616000	110912000	110.91	2	12
multi_4	7177	3488000	27824000	113584000	113.58	2	12
multi_5	5762	15408000	15168000	117344000	117.34	2	12
multi_6	6371	19840000	19728000	126224000	126.22	2	13
multi_7	5794	9712000	14400000	126304000	126.30	2	13
multi_8	6042	15776000	15536000	126960000	126.96	2	13
multi_9	5740	15984000	15808000	128320000	128.32	3	13
delayed	12568	22256000	30560000	230672000	230.67	4	24

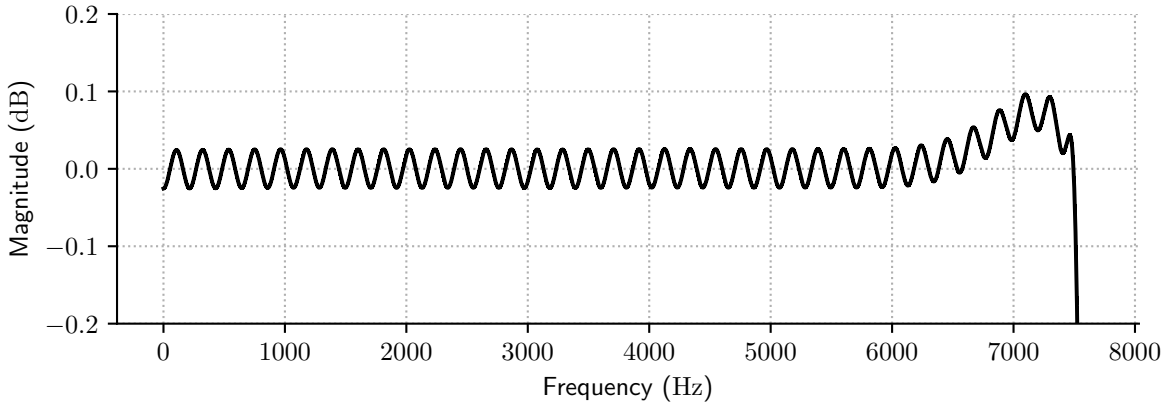
Table 5.1: Comparison of the proposed *delayed decimation filter* and the multi-stage decimation filters found in Chapter 3 by optimization algorithm.



(a)



(b)



(c)

Figure 5.13: Magnitude (a) and phase (b) frequency spectrum of the *delayed decimation filter*. (c) Passband ripple frequency spectrum.

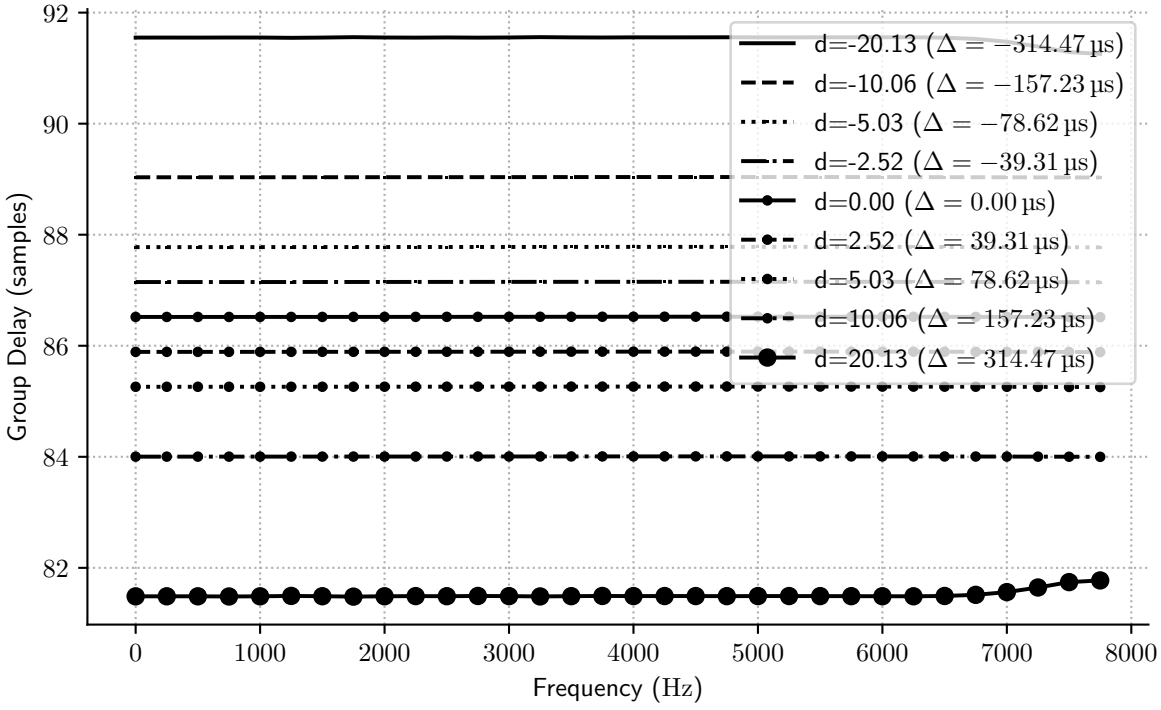


Figure 5.14: *Delayed decimation filter* group delay.

	S_z	S_*	S_+	S_o	f_{cpu} (MHz)	T^+_{FPGA} (-)	T^+_{lp} (-)
0	138	15680000	15616000	15616000	15.62	1	2
1	552	1536000	6144000	39936000	39.94	1	4
2	2714	0	3776000	3776000	3.78	1	1
3	9164	5040000	5024000	171344000	171.34	3	18

Table 5.2: *Delayed decimation filter* resource requirements breakdown. First row corresponds to the Lth-band filter stage, the second and third ones are to the $B_{N,K,d}(z)$ and $A_n(z)$ parts of the Samadi filter respectively, and the last one to the equiripple filter.

	Value	Unit
beamformer's storage requirement (S^z_{bf})	39478	bit
beamformer's number of multiplications per second (S^*_{bf})	6.9624e+08	MPS
beamformer's number of additions per second (S^+_{bf})	8.81696e+08	APS
beamformer's total number of additions per second (S^o_{bf})	2.45858e+09	APS
estimated minimum frequency in a processor (f_{cpu})	2458.58	MHz
estimated number of adders in an FPGA running at 64 MHz (T^+_{FPGA})	39	-
estimated number of adders in a VLSI circuit running at 10 MHz (T^+_{lp})	246	-

Table 5.3: Required resources to implement a beamformer using 40 shared *delayed decimation filters*.

	S^{z}_{dec} (bit)	S^{*}_{dec} (MPS)	S^{+}_{dec} (APS)	S^{o}_{dec} (APS)	f_{cpu} (MHz)	T^{+}_{FPGA} (-)	T^{+}_{lp} (-)
delayed_bf	39478	696240000	881696000	2458576000	2458.58	39	246
pcm_single_direct	1529520	1165886080000	2331648624000	3497548784000	3497548.78	54650	349755
pcm_single_eff	1529520	6072960000	12144624000	18231664000	18231.66	285	1824
pcm_single_memsav	27360	12145280000	12144624000	24303984000	24303.98	380	2431
pcm_multi_0	210040	1213440000	1211504000	4184944000	4184.94	66	419
pcm_multi_1	238280	770560000	766064000	4435184000	4435.18	70	444
pcm_multi_2	318160	160000000	785264000	4451824000	4451.82	70	446
pcm_multi_3	318160	160000000	785264000	4451824000	4451.82	70	446
pcm_multi_4	298600	140160000	1113584000	4558704000	4558.70	72	456
pcm_multi_5	242000	616960000	607344000	4709104000	4709.10	74	471
pcm_multi_6	266360	794240000	789744000	5064304000	5064.30	80	507
pcm_multi_7	243280	389120000	576624000	5067504000	5067.50	80	507
pcm_multi_8	253200	631680000	622064000	5093744000	5093.74	80	510
pcm_multi_9	241120	640000000	632944000	5148144000	5148.14	81	515
pdm_single_direct	115310	29270016000	58411008000	90384384000	90384.38	1413	9039
pdm_single_eff	115310	274688000	423408000	3401456000	3401.46	54	341
pdm_single_memsav	77756	426496000	423408000	3553264000	3553.26	56	356
pdm_multi_0	82323	153200000	150080000	3050288000	3050.29	48	306
pdm_multi_1	83029	142128000	138944000	3056544000	3056.54	48	306
pdm_multi_2	85026	126864000	139424000	3056960000	3056.96	48	306
pdm_multi_3	85026	126864000	139424000	3056960000	3056.96	48	306
pdm_multi_4	84537	126368000	147632000	3059632000	3059.63	48	306
pdm_multi_5	83122	138288000	134976000	3063392000	3063.39	48	307
pdm_multi_6	83731	142720000	139536000	3072272000	3072.27	49	308
pdm_multi_7	83154	132592000	134208000	3072352000	3072.35	49	308
pdm_multi_8	83402	138656000	135344000	3073008000	3073.01	49	308
pdm_multi_9	83100	138864000	135616000	3074368000	3074.37	49	308

Table 5.4: Comparison of the proposed beamformer based on *delayed decimation filter* and PDM and PCM domain beamformers discussed in Chapter 4.

Chapter 6

Conclusion

The main objective of this work was to find out an efficient way to implement a beamformer that uses PDM-mics. To achieve this objective, this work explored three approaches: to determine an optimum decimation filter, to perform the beamforming at PDM domain and to determine an optimum beamformer architecture that merges decimation and delay operations.

Determine an optimum decimation filter

This first approach (Chapter 3) required, at first, a review of the state-of-the-art filter design methods like equiripple, Lth-band and CIC filters. During this review, it was realized an opportunity to reduce the number of adders of CIC compensators at certain conditions ($\delta_p \geq 0.1\,\text{dB}$,$c \leq 1/4$,$R > 5$) just changing the coefficients of the state-of-the-art architecture, so a new set of coefficients for this condition was proposed. In the end, due to the fact that all discussed multiplierless CIC compensator design methods (summarized in Table 3.2) require less adders at different conditions, it is not concluded that one of them is the best, but that they are complementary to each other.

Once those filter design methods were reviewed, due to the prohibitive required resources to implement a beamformer using these single-stage basic structures, it was realized that a multi-stage approach is required. So, multi-stage multirate filter design was discussed such that, given a overall filter specification, the individual frequency ranges and ripples constraints of each stage were derived.

Finally, using the results of this single-stage and multi-stage filter design review, it was proposed an algorithm to calculate the structure of a multirate filter (Algorithm 2), and specifically, of a decimation filter minimizing the total number of additions per second (S_o). This algorithm was applied to calculate decimation filter structures with specification as listed in Table 1.1. From the algorithm results it is concluded that for the given specification, that considers an PDM input, the most efficient architecture will be a two-stages decimation filter (*multi_0*) with a first Lth-band stage and a second equiripple

filter stage, decimation 96 and 2 respectively. It can be concluded also that, as it is usually *de facto* assumed, a CIC filter-based architecture is not the most efficient alternative if implementation resources reduction is the objective.

Perform beamforming at PDM domain

At first, this second approach (Chapter 4) required us to discuss the state-of-the-art beamformer implementation methods. These state-of-the-art implementation methods assume that the signals incoming from the microphones are already in PCM domain and perform the beamforming in time or frequency domain. So, those time and frequency domain methods were further discussed in the context of use PDM-mics as sensors.

Once the state-of-the-art methods were reviewed, methods to do beamforming directly in PDM signals rather than PCM signals were proposed. As these methods do not require to convert all signals incoming from PDM-mics to PCM to do the beamforming, it was shown that they allow us to get rid of many decimation filters and, consequently, they require less storage and computational resources for their implementation.

Once the state-of-the-art and proposed beamforming implementations were discussed, the resources to implement a beamformer with specifications as listed in Table 1.2 and using the most efficient decimation filter found in Chapter 3 (*multi_0*) were calculated for each beamforming implementation (Table 4.6). The results showed that frequency domain implementations are not efficient and require a lot of computational resources because of the FFT blocks, as they require $O(N \log_2 N)$ operations and more memory to bufferize results. It is also shown that for the given conditions the state-of-the-art *discrete-time beamformer* and the proposed *discrete-time bitstream beamformer* are the most resource-efficient architectures.

The implementation resources of the state-of-the-art *discrete-time beamformer* and the proposed *discrete-time bitstream beamformer* were calculated using single-stage and multi-stage decimation filters calculated by optimization algorithm (Algorithm 2) in Chapter 3 for a beamformer specification as listed in Table 1.2. In Table 6.1 is shown the two most efficient state-of-the-art beamformer implementations (prefix *pcm_*) and the two most efficient proposed beamformer implementations (prefix *pdm_*) found by this analysis.

Finally, it is concluded that for the given beamformer and filtering specifications the *pcm_single_memsav* implementation will require less storage than any other method, but it will require also more additions per second than any other one. Discarding the *pcm_single_memsav* beamformer because of its high computational requirements, it is reasonable to conclude that from these 4 methods, the beamformers at PDM domain (*pdm_*) are the most efficient. Also from the comparison of these two ones, it can be inferred that the *pdm_multi_0* one is more suitable for software implementations, and

the *pdm_single_memsav* one could be more suitable for hardware implementation (FPGA or VLSI) as it requires less storage.

Determine an optimum beamformer architecture that merges decimation and delay operations

As is required a structure that performs filtering and delaying at the same time, the best candidate was the Samadi filter because of its capability to regulate its group delay without adding any delay element, just changing its coefficients. So, at first, we reviewed the Samadi filter properties and implementation structures (Chapter 5). As this filter class was not used in the state-of-the-art as a decimation filter, Algorithm 3 was proposed to calculate its parameters N and K based on a given decimation filter specification and a required group delay (d).

Once the Samadi filter was further analyzed, we proposed a decimation filter based on it called *delayed decimation filter*. It is a multi-stage filter with a Samadi filter stage and a last equiripple stage. Then, the design considerations and resource implementation expressions of this filter were further discussed. This *delayed decimation filter* was decomposed in its binomial components and it was used as the base of a novel DAS beamformer implementation. This proposed implementation is based on sharing common parts of the each *delayed decimation filter* that do not depend on the desired delay (Δ_m).

Also, a *delayed decimation filter* was designed with specification as listed in Table 1.1. This decimation filter was compared with the optimum decimation filter found in Chapter 3, but it was found that it is not efficient in comparison to them as it requires more implementation resources working as a standalone filter. Then this decimation filter was used to implement a DAS beamformer whose resource requirements are shown in Table 6.1 as *delayed decimation filter-based beamformer*. The resource requirements of most efficient beamformers architectures studied on this thesis are also summarized in this table.

This table shows that the proposed implementations at PDM domain (*pdm_*) and using *delayed decimation filters* are more efficient that the conventional ones regarding required area and number of adders/multipliers. This table also shows that the proposed beamformer based on the Samadi filter is the second most storage efficient option (S^z_{bf}), only after the single-stage implementation using polyphase memory-saving *pdm_single_memsav*. However, again due to the fact that *pdm_single_memsav* is computationally expensive, we could say that practically the most efficient beamformer is the one based on the proposed *delayed decimation filters*. In other words, the *delayed_bf* implementation provides the best trade-off between storage and computational resources and it would be a best choice

	Beamformer's storage requirement (S^z_{bf}) in bit	Beamformer's total number of additions per second (S^o_{bf}) in APS	Estimated minimum frequency in a processor (f_{cpu}) in MHz	Estimated number of adders in an FPGA running at 64 MHz (T^+_{FPGA})	Estimated number of adders in a VLSI circuit running at 10 MHz (T^+_{lp})
Delayed decimation filter-based beamformer (*delayed_bf*)	39478	2458576000	2458.58	39	246
Discrete-time beamformer using a multi-stage decimation filter (*pcm_multi_0*)	210040	4184944000	4184.94	66	419
Discrete-time beamformer using a single-stage memory saving decimation filter (*pcm_single_memsav*)	27360	24303984000	24303.98	380	2431
Discrete-time beamformer using a multi-stage decimation filter at PDM domain (*pdm_multi_0*)	82323	3050288000	3050.29	48	306
Discrete-time beamformer using a single-stage memory saving decimation filter at PDM domain (*pdm_single_memsav*)	77756	3553264000	3553.26	56	356

Table 6.1: Comparison of required resources to implement a 40 PDM-mic DAS beamformer with specifications listed in Table 1.1 and 1.2.

to be used in low-power consumption applications where storage and computational rate are both critical.

Finally, it is important to remark that this work focused in exploring and proposing new methods to implement beamformers efficiently without taking into account the beamformer detection efficiency, as this is considered transparent to its implementation.

Future work

The results of this work show the efficiency of using beamformers based on Samadi filters applied to broadband audio signals. A future work could explore the efficiency of the same beamformer structure in narrow band signals, as in this case flatness is usually not a required, it could be possible to reduce implementation resources even more.

Also, as it has been demonstrated that the binomial structure of Samadi filters are identical to the wavelet filters proposed by Daubechies [53], and as these structures are suitable for multi-resolution signal decomposition and coding applications; the proposed decimation filter based on Samadi filters could be used in efficient implementations of array processing algorithms in the wavelet domain that uses $\Sigma\Delta$M data as required for medical applications, for example.

Bibliography

[1] G. Jovanovic-Dolecek, R. Baez, G. Salgado, and J. Rosa, "Novel multiplierless wideband comb compensator with high compensation capability," *Circuits, Systems, and Signal Processing*, vol. 36, 08 2016.

[2] G. J. Dolecek and S. K. Mitra, "Simple method for compensation of cic decimation filter," *Electronics Letters*, vol. 44, pp. 1162–1163, Sep. 2008.

[3] G. Jovanovic-Dolecek and A. Fernandez-Vazquez, "Trigonometrical approach to design a simple wideband comb compensator," *AEU - International Journal of Electronics and Communications*, vol. 68, 01 2013.

[4] H. Krim and M. Viberg, "Two decades of array signal processing research: the parametric approach," *IEEE Signal Processing Magazine*, vol. 13, pp. 67–94, Jul 1996.

[5] G. W. Elko, *Future Directions for Microphone Arrays*, pp. 383–387. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[6] D. Van Compernolle, *Future Directions in Microphone Array Processing*, pp. 389–394. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[7] R. Lawes, *MEMS Cost Analysis: From Laboratory to Industry.* Pan Stanford, 2014.

[8] S. Carbajal and B. Masiero, "Microphone array processing of pulse-density modulated bitstreams," in *XXVIII Encontro da SOBRAC*, March 2020.

[9] S. Samadi, A. Nishihara, and H. Iwakura, "Universal maximally flat lowpass fir systems," *IEEE Transactions on Signal Processing*, vol. 48, pp. 1956–1964, July 2000.

[10] P. Malcovati and A. Baschirotto, "The evolution of integrated interfaces for mems microphones," *Micromachines*, vol. 9, p. 323, 06 2018.

[11] S. Park and Motorola, *Motorola Digital Signal Processors: Principles of Sigma-delta Modulation for Analog-to-digital Converters.* Motorola, 1993.

[12] J. de la Rosa and R. Río, *CMOS Sigma-Delta Converters: Practical Design Guide.* Wiley - IEEE, Wiley, 2013.

[13] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed., 2009.

[14] L. Milic, *Multirate Filtering for Digital Signal Processing: MATLAB Applications: MATLAB Applications*. Premier reference source, Information Science Reference, 2009.

[15] B. Metzler, *Audio Measurement Handbook*. Audio Precision, 1993.

[16] H. Suzuki, S. Morita, and T. Shindo, "On the perception of phase distortion," *Journal of the Audio Engineering Society*, vol. 28, pp. 570–574, September 1980.

[17] M. Bellanger, G. Bonnerot, and M. Coudreuse, "Digital filtering by polyphase network:application to sample-rate alteration and filter banks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, pp. 109–114, April 1976.

[18] N. Fliege, *Multirate digital signal processing: multirate systems, filter banks, wavelets*. Wiley, 1994.

[19] R. Crochiere and L. Rabiner, "Optimum fir digital filter implementations for decimation, interpolation, and narrow-band filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, pp. 444–456, October 1975.

[20] R. Crochiere and L. Rabiner, *Multirate Digital Signal Processing*. Prentice-Hall Signal Processing Series: Advanced monographs, Prentice-Hall, 1983.

[21] L. Rabiner and B. Gold, *Theory and application of digital signal processing*. Prentice-Hall signal processing series, Prentice-Hall, 1975.

[22] E. Hofstetter, A. V. Oppenheim, and J. Siegel, "A new technique for the design of non-recursive digital filters," in *5th Annual Princeton Conference on Information Sciences and Systems*, March 1971.

[23] T. Parks and J. McClellan, "Chebyshev approximation for nonrecursive digital filters with linear phase," *IEEE Transactions on Circuit Theory*, vol. 19, pp. 189–194, March 1972.

[24] J. McClellan, T. Parks, and L. Rabiner, "A computer program for designing optimum fir linear phase digital filters," *IEEE Transactions on Audio and Electroacoustics*, vol. 21, pp. 506–526, December 1973.

[25] J. McClellan and T. Parks, "A unified approach to the design of optimum fir linear-phase digital filters," *IEEE Transactions on Circuit Theory*, vol. 20, pp. 697–701, November 1973.

[26] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python," *arXiv e-prints*, p. arXiv:1907.10121, Jul 2019.

[27] O. Herrmann, L. R. Rabiner, and D. S. K. Chan, "Practical design rules for optimum finite impulse response low-pass digital filters," *The Bell System Technical Journal*, vol. 52, pp. 769–799, July 1973.

[28] F. Mintzer, "On half-band, third-band, and nth-band fir filters and their design," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 30, pp. 734–738, October 1982.

[29] E. Hogenauer, "An economical class of digital filters for decimation and interpolation," vol. 29, pp. 155 – 162, 05 1981.

[30] A. Fernandez-Vazquez and G. Jovanovic Dolecek, "A general method to design gcf compensation filter," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, pp. 409–413, May 2009.

[31] A. Fernandez-Vazquez and G. Jovanovic-Dolecek, "Maximally flat cic compensation filter: Design and multiplierless implementation," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 59, pp. 113–117, 02 2012.

[32] G. J. Dolecek, "Simple wideband cic compensator," *Electronics Letters*, vol. 45, pp. 1270–1272, November 2009.

[33] G. Jovanovic-Dolecek and f. harris, "Design of wideband cic compensator filter for a digital if receiver," *Digital Signal Processing*, vol. 19, pp. 827–837, 09 2009.

[34] S. Kim, W.-C. Lee, S. Ahn, and S. Choi, "Design of cic roll-off compensation filter in a w-cdma digital if receiver," *Digital Signal Processing*, vol. 16, pp. 846–854, 11 2006.

[35] G. Molnar and M. Vucic, "Closed-form design of cic compensators based on maximally flat error criterion," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, pp. 926–930, Dec 2011.

[36] M. G. Pecotic, G. Molnar, and M. Vucic, "Design of cic compensators with spt coefficients based on interval analysis," in *2012 Proceedings of the 35th International Convention MIPRO*, pp. 123–128, May 2012.

[37] D. E. T. Romero and G. J. Dolecek, "Application of amplitude transformation for compensation of comb decimation filters," *Electronics Letters*, vol. 49, pp. 985–987, Aug 2013.

[38] K. S. Yeung and S. C. Chan, "The design and multiplier-less realization of software radio receivers with reduced system delay," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, pp. 2444–2459, Dec 2004.

[39] D. E. T. Romero and M. G. C. Jimenez, "Efficient wide-band droop compensation for cic filters: ad hoc and reconfigurable fir architectures," *Electronics Letters*, vol. 53, no. 4, pp. 228–229, 2017.

[40] L. Xu, W. Yang, and H. Tian, "Design of wideband cic compensator based on particle swarm optimization," *Circuits, Systems, and Signal Processing*, vol. 38, pp. 1833–1846, Apr 2019.

[41] D. Lamb, *Efficient algorithms and hardware structures for fractional delay filtering and sample rate conversion*. PhD thesis, Universidade de Sao Paulo, 2016.

[42] R. W. Schafer and L. R. Rabiner, "A digital signal processing approach to interpolation," *Proceedings of the IEEE*, vol. 61, pp. 692–702, June 1973.

[43] J. Tiete, F. Domínguez, B. Silva, L. Segers, K. Steenhaut, and A. Touhafi, "Soundcompass: A distributed mems microphone array-based sensor for sound source localization," *Sensors*, vol. 14, p. 1918–1949, Jan 2014.

[44] K. Youngkey, K. Jungoo, and L. Myunghan, "Developing beam-forming devices to detect squeak and rattle sources by using fpga," in *Inter-noise*, pp. 1–6, 2014.

[45] O. Herrmann, "On the approximation problem in nonrecursive digital filter design," *IEEE Transactions on Circuit Theory*, vol. 18, pp. 411–413, May 1971.

[46] J. A. Miller, "Maximally flat nonrecursive digital filters," *Electronics Letters*, vol. 8, pp. 157–158, March 1972.

[47] M. F. Fahmy, "Maximally flat non-recursive digital filters," *International Journal of Circuit Theory and Applications*, vol. 4, no. 3, pp. 311–313, 1976.

[48] L. Rajagpoal and S. D. Roy, "Design of maximally-flat fir filters using the bernstein polynomial," *IEEE Transactions on Circuits and Systems*, vol. 34, pp. 1587–1590, December 1987.

[49] H. Baher, "Fir digital filters with simultaneous conditions on amplitude and delay," *Electronics Letters*, vol. 18, pp. 296–297, April 1982.

[50] S. Samadi and A. Nishihara, "Concise representation and cellular structure for universal maximally flat fir filters," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, vol. 4, pp. IV–IV, May 2003.

[51] P. Vaidyanathan, "On maximally-flat linear-phase fir filters," *IEEE Transactions on Circuits and Systems*, vol. 31, pp. 830–832, Sep. 1984.

[52] R. Haddad, "A class of orthogonal nonrecursive binomial filters," *IEEE Transactions on Audio and Electroacoustics*, vol. 19, pp. 296–304, December 1971.

[53] A. Akansu, R. Haddad, and H. Caglar, "The binomial qmf-wavelet transform for multiresolution signal decomposition," *Signal Processing, IEEE Transactions on*, vol. 41, pp. 13–, 02 1993.